François Monteil

Come usare il tuo COMMODORE 64

Editrice Il Rostro

Come usare il tuo Commodore 64

1. BASIC, grafica e suoni

Pubblicazioni di microinformatica della Editrice Il Rostro

H.J. Sacht - *I personal computer. Vademecum per chi vuole conoscerli, usarli, comprarli*

B.A. Artwick -*Interfacciamento dei microcomputer. Metodi e dispositivi*

F. Monteil - *Come usare il tuo Commodore 64*

F. e M. G. Monteil - *Come usare il tuo VIC 20*

J.Y. Astier - *Come usare il tuo Apple II*

Bernd Pol - *Come programmare in BASIC*

V. Cappelli - *Guida breve alla programmazione in BASIC*

R. Schomberg - *Il BASIC del tuo personal*

K.J. Schmidt e G. Renner - *Sistemi operativi per microcomputer: CP/M-CDOS-DOS*

I.R. Wilson e A.M. Addyman - *Introduzione al Pascal*

François Monteil

Come usare il tuo Commodore 64

1. BASIC, grafica e suoni

Editrice Il Rostro

Indice

Prefazione

Quest'opera si rivolge a quanti possiedono o intendono acquistare un computer Commodore 64.
Questa macchina offre un gran numero di possibilità di utilizzo cosicché è parso ragionevole suddividere in due parti l'opera "Come usare il tuo Commodore 64". La prima di esse affronta i seguenti argomenti:

— il BASIC;
— le diverse modalità di rappresentazione sullo schermo;
— il sintetizzatore di suoni.

La seconda parte è consacrata al linguaggio macchina, alle funzioni di ingresso/uscita e alle periferiche.
Questo libro non si pone in alternativa con il manuale di impiego fornito dalla Commodore insieme al calcolatore. Il capitolo sul BASIC descrive le particolarità di ciascuna istruzione o di ciascun comando e fornisce numerosi esempi di impiego. Il capitolo sul controllore video descrive in dettaglio i diversi modi di visualizzazione. Dopo aver letto questo capitolo potrete accedere a un generatore di caratteri personalizzato, alla grafica ad alta risoluzione e alla programmazione degli SPRITE che costituiscono l'originalità di questa macchina.
Il capitolo sulla sintesi dei suoni permetterà di imparare a programmare suoni e musiche.
Mentre la prima parte si rivolge a tutti con esempi scritti in BASIC, la seconda parte interessa quanti vogliono andare più lontano e sfruttare a fondo tutte le possibilità del calcolatore utilizzando la programmazione in linguaggio Assembler.
Quest'opera non pretende in alcun modo di essere un corso di programmazione in BASIC o in Assembler. A questo scopo occorre fare riferimento ad altre opere* pubblicate su questo argomento o sul microprocessore 6510 che viene utilizzato nel Commodore 64.

---

* Il lettore interessato potrà consultare l'elenco dei volumi di microinformatica della Editrice Il Rostro che appare all'inizio del libro.

1
I primi contatti

Il Commodore 64 è forse il primo personal computer che offre una grande capacità di memoria (64 kbyte), possibilità grafiche e sonore stupefacenti, vaste possibilità di ampliamenti a un prezzo relativamente contenuto.
Detto questo, va però notato che, come del resto il VIC 20 venduto dal medesimo costruttore, il computer è fornito senza schermo di visualizzazione: questo pone alcuni problemi.
Il Commodore 64 disponibile in Italia permette di rappresentare testi e grafici a colori se collegato a un qualsiasi televisore a colori sistema PAL.
Al momento dell'acquisto però potrete sfruttare più soluzioni di visualizzazione:

— visualizzazione monocromatica in bianco e nero, perdendo così molte caratteristiche grafiche della macchina;
— visualizzazione a colori.

Se disponete di un monitor in bianco e nero (non un televisore) o di un televisore in bianco e nero con ingresso video (caso molto raro) disporrete di un'immagine monocromatica di buona qualità, non dovrete fare adattamenti, ma non disporrete però del suono (una caratteristica molto curata nel 64).
Per rimediare a ciò è però possibile collegarsi a un amplificatore e ad un altoparlante esterno attraverso la presa Video-Audio. Disporrete allora di un sintetizzatore musicale di qualità Hi-Fi, o quasi.

Per effettuare i collegamenti sia tra Commodore 64 e monitor sia tra Commodore 64 e amplificatore basterà utilizzare un cavo che parte dalla presa Video-Audio situata posteriormente alla macchina. Se invece disponete solo di un televisore in bianco e nero potete collegarlo direttamente al calcolatore e disporrete contemporaneamente del sonoro tramite l'altoparlante integrato nel televisore stesso. Sfortunatamente la qualità dell'immagine sarà mediocre e il vostro 64 perderà molto sul piano della grafica.
La maggior parte di voi vorrà però utilizzare il Commodore 64 con il televisore a colori. Nessun problema. Basterà collegare la macchina direttamente sulla presa d'antenna del televisore utilizzando il cavo fornito dalla casa costruttrice. Avrete contemporaneamente immagini e suoni.
Fatto il collegamento potrete accendere televisione e calcolatore. Dopo alcuni istanti durante i quali quest'ultimo effettua certe operazioni di inizializzazione vedrete apparire il messaggio:

```
**** CBM BASIC V2 ****
38911 BYTES FREE
```

e la scritta READY.

2
Il BASIC del Commodore 64

2.1 Introduzione

In questo capitolo ci proponiamo di descrivere tutte le istruzioni e i comandi BASIC presenti sul Commodore 64. Abbiamo scelto di illustrarli raggruppandoli in categorie affini per permettervi di utilizzarli più rapidamente e in modo più efficace. Numerosi esempi precisano il loro impiego. Detto questo, vogliamo sottolineare che questo capitolo non pretende di essere un corso di BASIC: per approfondire certi argomenti o certe istruzioni del BASIC potrete fare riferimento ad altre opere specializzate, come quelle pubblicate dall'editore di questo libro.

2.2 Le variabili

Il BASIC del 64 accetta tre tipi di variabili:

— tipo intero
— tipo reale
— tipo stringa.

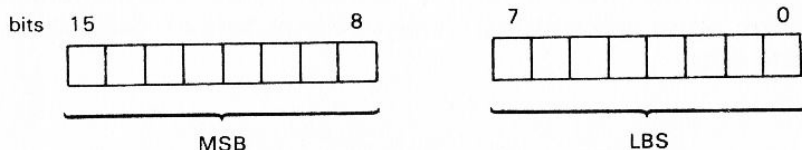
Ad ogni variabile è attribuito un nome formato da uno o da due caratteri alfabetici o numerici, di cui il primo deve obbligatoriamente

essere una lettera dell'alfabeto. Tuttavia si possono anche utilizzare nomi più lunghi, ma l'interprete BASIC della macchina non tiene conto che dei due primi caratteri. Occorre dunque fare attenzione a non usare nomi diversi con le stesse due lettere iniziali. Il tipo di una variabile viene dichiarato per mezzo di un carattere addizionale (% per il tipo intero e $ per il tipo stringa). Senza quest'ultimo il BASIC considera la variabile come di tipo reale.

Esempio:

A% e V1% sono intere
AC e B sono reali
A1$ e B$ sono stringhe di caratteri.

2.2.1 Variabili di tipo intero

Il Commodore 64 tratta numeri interi compresi tra —32768 e +32767. Sono dunque necessarie 2 parole di 8 bit per immagazzinare un intero in notazione in complemento a 2. Il byte più significativo (MSB) è contenuto nella prima e precede il byte meno significativo (LSB).

Il bit 15 di questa parola di 16 bit rappresenta il segno: se vale "zero" il numero intero è positivo; se vale "uno" il numero è negativo.

Ad esempio:

Per X = 255 si avrà MSB = 0 e LSB = 255
Per X = —255 si avrà MSB = 255 e LSB = 1
Per X = 32 767 si avrà MSB = 127 e LSB = 255
Per X = —32767 si avrà MSB = 128 e LSB = 0

Nel caso di un numero positivo si avrà:

$$\text{Numero} = a_{14}\, 2^{14} + a_{13}\, 2^{13} + a_{12}\, 2^{12} + \dots + a_1\, 2^1 + a_0$$

Nel caso invece di un numero negativo si avrà:

$$\text{Numero} = -(2^{15} - (a_{14}\, 2^{14} + a_{13}\, 2^{13} + \dots + a_1\, 2^1 + a_0))$$

2.2.2 Variabili di tipo reale

Nel Commodore 64 le variabili di tipo reale possono essere rappresentate per mezzo di un massimo di nove cifre. I calcoli sono effettuati con una precisione di dieci cifre significative, ma il risultato è troncato a nove cifre.
I numeri reali necessitano di 5 parole di 8 bit (byte) per essere immagazzinati in memoria con formato in virgola mobile. Per ogni numero reale sono necessarie tre informazioni:

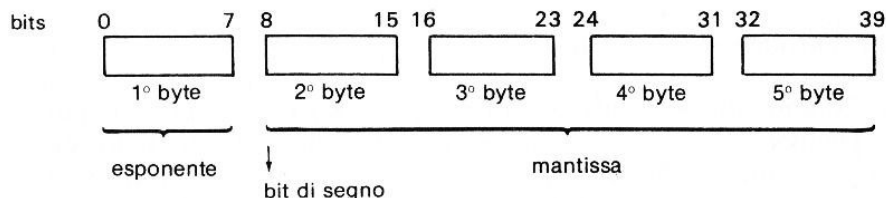
— mantissa (compresa tra 0,5 e 1 in valore assoluto)
— segno
— esponente

Numereremo da 0 a 39 i 40 bit corrispondenti ai 5 byte. Nel Commodore 64 il primo byte (bit da 0 a 7) rappresenta l'esponente in cifre binarie grazie a una rappresentazione a modulo 128 nel seguente modo:

128 per un esponente nullo
130 per un esponente uguale a 2
126 per un esponente uguale a —2
255 per un esponente uguale a 127
0 per un esponente uguale a —128.

Il bit 8 rappresenta il segno della mantissa: se vale "uno", il numero è negativo; se vale "zero", il numero è positivo.
I bit da 9 a 39 rappresentano la mantissa compresa tra 0 e 0.5. Viene sottinteso un trentaduesimo bit, che vale sempre "uno" e permette di avere una mantissa il cui valore è sempre compreso tra 0.5 e 1 (viene detta normalizzata):

Il valore assoluto del numero rappresentato da questi cinque byte è dato dalla formula:

$$|\text{Numero}| = (2^{-1} + a_9 2^{-2} + a_{10} 2^{-3} + a_{11} 2^{-4} + \dots + a_{38} 2^{-31} + a_{39} 2^{-32}) 2^{(\text{esponente} - 128)}$$

Ad esempio, con il numero binario:

10100001 11100010 00010100 11101000 10111010

10100001: esponente = 161

11100010 (primo bit): segno negativo

si ha:

$$\text{Mantissa} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-7} + 2^{-12} + 2^{-14} + 2^{-17} + 2^{-18} + 2^{-19} + 2^{-21} + 2^{-25} + 2^{-27} + 2^{-28} + 2^{-29} + 2^{-31} = 0.88313155$$

Da cui:

$$\text{Numero} = -0.88313155 \times 2^{161-128}$$

$$\text{Numero} = -7.5860422 \quad 10^{9}$$

Il valore più alto si ottiene quando tutti i bit sono uguali a "uno" (bit di segno = 0 o 1). Dunque:

$$-1.70141183 \text{ E } 38 \leqslant \text{Numero} \leqslant 1.70141183 \text{ E } 38$$

Analogamente il valore più basso si ottiene quando tutti i bit sono uguali a "zero". Dunque:

$$|\text{Numero}| \geqslant 2.93873588 \text{ E } -39$$

2.2.3 Variabili di tipo stringa

I differenti caratteri che costituiscono una stringa (catena) di caratteri sono immagazzinati in memoria per mezzo della loro codifica ASCII. Ad esempio, alla stringa A$="COMMODORE-64" corrispondono in memoria i codici 67, 79, 77, 77, 79, 82, 69, 45, 54, 52 (cfr. tavola in Appendice).
Una stringa può contenere al massimo 255 caratteri.

2.3 Gli operatori

Esistono tre tipi di operatori:

— operatori aritmetici,
— operatori di confronto,
— operatori logici.

2.3.1 Gli operatori aritmetici

Sono:

— addizione (+),
— sottrazione (—),
— moltiplicazione (*),
— divisione (/),
— elevamento a potenza (↑).

Questi diversi operatori sono ordinati in modo gerarchico (o di precedenza) tra di loro:

1) elevamento a potenza,
2) moltiplicazione o divisione,
3) addizione o sottrazione.

Ad esempio, 3.28 + 4 * 6 ↑ 2 sarà valutato nel modo seguente:

3.28 + (4*(6 ↑ 2)) = 147.28

2.3.2 Gli operatori di confronto

Sia con variabili numeriche sia con variabili di stringa si possono operare confronti per mezzo dei seguenti operatori:

— uguale (=),
— minore (<),
— maggiore (>),
— minore o uguale (<=),
— maggiore o uguale (>=),
— diverso (<>).

Nel caso di variabili di stringa, il Commodore 64 considera ogni carattere ad uno ad uno e confronta i codici ASCII corrispondenti.

2.3.3 Gli operatori logici

Gli operatori logici sono:

— "E" logico: AND,
— Negazione logica: NOT,
— "O" logico: OR.

I tre operatori lavorano solo su interi (o su variabili dichiarate di tipo intero) e le operazioni si effettuano bit per bit.
Diamo qui di seguito le tavole di verità di questi operatori.

AND	0	1
0	0	0
1	0	1

Dunque:
X AND 1 = X
e
X AND 0 = 0

OR

	0	1
0	0	1
1	1	1

Dunque:

X OR 1 = 1
e
X OR 0 = X

NOT fornisce invece il complemento della variabile considerata. Si avrà:

NOTX= $\overline{X}$ ($\overline{X}$= complemento di X)

Ad esempio:

36 AND 15 = 4

	MSB	LSB
36 è rappresentato da:	00000000	00100100
15 è rappresentato da:	00000000	00001111
Dunque 36 AND 15 è rappresentato da:	00000000	00000100
Analogamente si avrà: 36 OR 15 = 47 =	00000000	00101111
Come pure: NOT 36 = – 37 =	11111111	11011011

2.4 I comandi BASIC

Nei paragrafi che seguono descriveremo i diversi programmi BASIC disponibili sul Commodore 64.
Come potrete constatare i comandi e le istruzioni BASIC del Commodore 64 sono i medesimi di quelli del VIC 20.

NEW

Questo comando cancella il programma BASIC che si trova in memoria. In realtà esso inizializza nuovamente il puntatore che indica l'inizio della zona di immagazzinamento delle variabili (questa zona si trova subito dopo il programma BASIC). Su questo argomento ritorneremo più avanti.
Il comando NEW può essere utilizzato anche nel corso di un programma: esso provocherà la cancellazione di tutto quanto si trova in memoria al momento dell'esecuzione.

LIST

Questo comando provoca la visualizzazione sullo schermo, totale o parziale, del vostro programma in BASIC.
La sintassi è la seguente:

LIST [n° linea iniziale] — [n° linea finale]

Così:

LIST	: provoca la visualizzazione di tutto il programma
LIST 200	: mostra unicamente la linea 200
LIST 100—200	: mostra le linee da 100 a 200
LIST—200	: mostra il programma fino alla linea 200
LIST 200 —	: provoca la visualizzazione del programma a partire dalla linea 200.

Normalmente il programma è visualizzato sullo schermo ma è possibile, come vedremo, utilizzare in uscita altre periferiche (stampante, unità a disco) utilizzando l'istruzione CMD.

Un listing che scorre sullo schermo può essere rallentato premendo il tasto CTRL. Può essere arrestato schiacciando il tasto RUN/STOP.

RUN

Questo comando dà il via all'esecuzione di un programma.
La sintassi è la seguente:

RUN [n° linea]

Il comando RUN (da solo) permette il lancio del programma residente in memoria a partire dalla prima linea. RUN 100 fa partire invece il programma dalla linea 100.

CONT

Questo comando permette di rilanciare l'esecuzione di un programma che era stata fermata con una istruzione di STOP o con la pressione del tasto RUN/STOP.
Il programma continua, cominciando dall'indirizzo a cui era stato bloccato. Utilizzando congiuntamente l'istruzione STOP e il comando CONT è possibile una buona messa a punto dei programmi (debugging).
Infatti, così facendo, è possibile esaminare il valore delle variabili utilizzate dal programma al momento del suo arresto.

LOAD

Questo comando permette di caricare nella memoria RAM un programma precedentemente immagazzinato su disco o su cassetta.
La sintassi è la seguente:

LOAD ["nome del file"] [, n° della periferica] [, comando]

Esamineremo separatamente il caso di un caricamento a partire da una cassetta o da un disco.

1) Caricamento a partire da una cassetta
In questo caso il "nome del file" è facoltativo. Se viene omesso il 64 carica il primo file disponibile.

Il numero della periferica è 1, ma può essere omesso. Il funzionamento del comando è il seguente:

Si batte sulla tastiera:

```
LOAD "PROGRAMMA", 1, 1
```

o semplicemente

```
LOAD
```

Dopo aver premuto il tasto RETURN, il lettore di cassette va messo in posizione di lettura (PLAY).
Quando ha individuato il programma, il Commodore 64 emette il messaggio "FOUND" e premendo uno dei tasti CTRL, ←, SPACE o C=(logo Commodore) si procede al caricamento.
Normalmente il programma verrà caricato a partire dall'indirizzo 2048. Se si utilizza il comando 1 (ad esempio: LOAD "PROGRAMMA", 1, 1) il programma denominato PROGRAMMA sarà caricato al medesimo indirizzo in cui si trovava quando è stato memorizzato con un comando SAVE.

2) Caricamento a partire da un disco
In questo caso né il "nome del file" né il numero della periferica (8) possono essere omessi. Si scriverà ad esempio:

```
LOAD "PROGRAMMA", 8
```
oppure:
```
    LOAD "*", 8
```

che provoca il caricamento del primo file presente sulla "Directory" del dischetto utilizzato. Analogamente a quanto avviene per il caricamento da cassetta, un comando 1 permette il caricamento al medesimo indirizzo in cui si trovava quando è stato memorizzato con un SAVE.

SAVE

Inversamente dal comando LOAD, il comando SAVE permette di

memorizzare su cassetta o su disco un programma BASIC che si trova nella memoria RAM del calcolatore.
La sintassi è la seguente:

SAVE ["nome del file"] [, n° di periferica] [, comando]

Va notato che il programma che si trova nella memoria del calcolatore non viene toccato dal comando SAVE. Come nel caso LOAD, bisogna distinguere due possibilità: cassetta o disco.

1) Memorizzazione su cassetta
In questo caso il "nome del file" può essere omesso e il programma sarà memorizzato su cassetta senza alcun nome. Il numero della periferica è ancora 1, ma può essere ugualmente omesso. Il comando è opzionale e può prendere 3 valori:

— *comando = 1*: indica al Commodore 64 che il programma "salvato" non dovrà cambiare il suo posto in memoria al momento di un futuro caricamento;
— *comando = 2*: utile unicamente per il lettore di cassette, questo comando provoca la scrittura di un segnale di fine nastro subito dopo il programma "salvato"; questo permette, al momento del caricamento, di evitare di andare a leggere al di là dell'ultima informazione relativa al programma;
— *comando = 3*: combina le funzioni dei due comandi precedenti.

Ad esempio:

```
SAVE
SAVE "PROGRAMMA", 1
SAVE "PROGRAMMA", 1, 2
```

2) Memorizzazione su disco
Come nel caso del comando LOAD né il "nome del file" né il numero della periferica possono essere omessi.

Ad esempio:

```
SAVE "PROGRAMMA", 8
SAVE "PROGRAMMA", 8, 1
```

Nel caso di una memorizzazione su disco si può utilizzare solo il comando 1, che ha un funzionamento analogo a quello della cassetta.

VERIFY

Questo comando permette di confrontare il programma presente nella memoria RAM con quello presente sulla cassetta o sul disco che porta lo stesso nome specificato nel comando.
La sintassi è la seguente:

VERIFY ["nome del file"] [, n° di periferica]

In generale questo comando viene utilizzato subito dopo il SAVE. Nel caso di un lettore di cassette, il nome del file e il numero della periferica possono essere omessi. Invece nel caso di un disco devono essere specificati.
Si scriverà ad esempio:

VERIFY
oppure:
VERIFY "PROGRAMMA", 1 (cassetta)
oppure:
VERIFY "PROGRAMMA", 8 (disco)

Se i due programmi a confronto non sono rigorosamente identici apparirà sullo schermo il messaggio:

? VERIFY ERROR

2.5 Le istruzioni BASIC

2.5.1 Le istruzioni matematiche

Possiamo raggrupparle nella tavola seguente:

Nome	Funzione	Condizioni sull'argomento
ABS(X)	Valore assoluto di X	Nessuna
ATN(X)	Arcotangente di X	Nessuna
COS(X)	Coseno di X	Nessuna
EXP(X)	Esponenziale di X	$X < 88.0296919$
INT(X)	Parte intera di X	Nessuna
LOG(X)	Logaritmo di X	$X > 0$
RND(X)	Generazione di numero casuale compreso tra 0 e 1	Nessuna
SGN(X)	Signum X: +1 se X>0, 0 se X=0 —1 se X 0	Nessuna
SIN(X)	Seno di X	Nessuna
SQR(X)	Radice quadrata di X	$X > = 0$
TANX(X)	Tangente di X	$X \neq \frac{\pi}{2} + k\,\pi$(k intero)

2.5.2 Le istruzioni logiche

Queste istruzioni le abbiamo incontrate nel paragrafo consacrato agli operatori BASIC.

Si tratta delle istruzioni:

AND
OR
NOT

Esempi:

Z = X AND Y
Z = NOT X
Z = X OR Y

2.5.3 Le istruzioni di stringa

ASC

Questa istruzione permette di estrarre il codice ASCII del primo carattere di una stringa X$.
La sintassi è la seguente:

```
ASC(X$)
```

Esempio:

```
10 X$ = "COMMODORE-64"
20 PRINT ASC(X$)
```

e il risultato mostrato sullo schermo sarà dunque 67.
Va notato che il numero estratto da un'istruzione ASC è sempre compreso tra 0 e 255.

CHR$

Questa istruzione realizza l'operazione inversa di ASC. Essa permette di trasformare un codice ASCII nel carattere equivalente.
La sintassi è la seguente:

```
CHR$(X)
```

Scrivendo, ad esempio, PRINT CHR$(13) si provoca un RETURN, cioè un "a capo" nella visualizzazione sullo schermo.
In Appendice forniamo una lista dei caratteri disponibili sul Commodore 64 e dei codici associati.

LEFT$

Questa istruzione permette di estrarre i primi I caratteri di una stringa X$.
La sintassi è la seguente:

```
LEFT$(X$,I)
```

Ad esempio, se si batte:

```
PRINT LEFT$(X$,2)
```

con X$= "COMMODORE-64", si ottiene la stringa di caratteri "CO"
Va notato che I deve essere sempre inferiore a 255 e che se I supera la lunghezza della stringa X$, quest'ultima sarà rappresentata per intero.

LEN

Questa istruzione calcola la lunghezza di una stringa.
La sintassi è la seguente:

```
LEN(X$)
```

Ad esempio:

```
10 X$ = "COMMODORE-64"
20 PRINT LEN(X$)
```

fornirà il risultato: 12.

MID$

Questa istruzione permette di estrarre J caratteri di una stringa a partire dal carattere I-esimo.
La sintassi è la seguente:

```
MID$(X$,I,J)
```

Ad esempio:

```
10 X$ = "COMMODORE-64, VIC-20"
20 PRINT MID$(X$,14,3)
```

darà come risultato la stringa "VIC".
Va notato che J e J devono essere minori di 255.
Se I>LEN (X$) o se J = 0 il risultato è una stringa vuota (" ")

RIGHT$

Questa istruzione permette di estrarre i primi I caratteri di una stringa.
La sintassi è la seguente:

```
RIGHT$(X$,I)
```

Ad esempio:

```
10 X$ = "COMMODORE-64, VIC-20"
20 PRINT RIGHT$(X$,6)
```

darà come risultato la stringa "VIC-20".

STR$

Questa istruzione permette di trasformare un numero in una stringa i cui caratteri sono quelli utilizzati per la sua visualizzazione sullo schermo.
La sintassi è la seguente:

```
STR$(X)
```

Ad esempio:

```
10 X = 1.25 E 3
20 Y$ = STR$(X)
30 PRINT Y$
```

darà il risultato 12500

VAL

Questa istruzione realizza l'operazione inversa di STR$. Essa fornisce il valore numerico che rappresenta il dato contenuto nella stringa. La sintassi è la seguente:

```
VAL(X$)
```

esempio:

```
10 INPUT X$
20 IF VAL(X$) > 75 001 AND VAL(X$) < 75 020
THEN PRINT "PARIS"; SPC(2); RIGHT$(X$,2)
```

Va notato che se il primo carattere della stringa non è una cifra oppure un segno + o —, il risultato ottenuto è 0.

2.5.4 Le istruzioni di salto

GOTO

Si tratta della classica istruzione BASIC per il salto incondizionato.

La sintassi è la seguente:

```
GOTO n° di linea
```

Ad esempio:

```
10 GOTO 100
```

provoca un salto alla linea 100 quando il programma esegue la linea numero 10.

GOSUB

E' l'istruzione di chiamata di un sottoprogramma.

La sintassi è la seguente:

```
GOSUB n° di linea
```

Ad esempio:

```
10 GOSUB 100
```

provoca una chiamata al sottoprogramma che comincia con la linea 100 (e termina con un'istruzione RETURN), quando il programma esegue la linea numero 10.
Forniamo un esempio di impiego di tale istruzione:

```
 10 REM PROGRAMMA PRINCIPALE
 20 DIM A(99)
 30 GOSUB 100
 40 END
100 REM SOTTOPROGRAMMA
110 FOR I = 0 TO 99
120 A(I) = I*I: PRINT A(I)
130 NEXT I
140 RETURN
```

END

Questa istruzione segna la fine di un programma BASIC e provoca la visualizzazione del READY. Va notato che questa istruzione è inutile nel caso del Commodore 64, poiché il programma si ferma automaticamente dopo aver eseguito l'ultima linea di istruzioni.

RETURN

Come già detto prima, è l'istruzione che provoca il ritorno dal sottoprogramma al programma principale.

STOP

Questa istruzione provoca l'arresto del programma in BASIC e la visualizzazione della scritta BREAK IN LINE N, dove N indica la linea che contiene l'istruzione STOP.
Questa istruzione non distrugge le variabili e non provoca la chiusura dei file che sono stati aperti al momento dell'esecuzione del programma. Per far ripartire il programma dalla linea in cui si trova lo STOP,

basta battere sulla tastiera il comando CONT, di cui ci siamo già occupati

ON

Questa istruzione va utilizzata insieme con le istruzioni GOTO e GOSUB.

Essa permette salti o chiamate di sottoprogrammi eseguiti in funzione del risultato di certi "calcoli" (come vedremo tra poco).

La sintassi è la seguente:

```
ON variabile GOTO n° di linea [, n° di linea 2, n° di linea 3,...., n° di linea n]
ON VARIABILE GOSUB n° di linea 1 [, n° di linea 2, n° di linea 3,..., n° di linea n]
```

In dipendenza dal valore variabile, il programma esegue il salto ed una certa linea o chiama il sottoprogramma che inizia una certa linea. Se la variabile è uguale a 1, si verifica il salto al primo numero di linea indicato; se la variabile è uguale a 2, si salta al secondo numero di linea; e così via...
Il programma seguente illustra il funzionamento di ON...GOTO:

```
10 INPUT A
20 ON A GOTO 30, 50, 70
30 PRINT "LINEA 30"
40 GOTO 10
50 PRINT "LINEA 50"
60 GOTO 10
70 PRINT "LINEA 70"
80 GOTO 10
90 END
```

Il funzionamento di ON...GOSUB è completamente analogo.

IF...THEN

Si tratta dell'istruzione di salto condizionato al numero di linea indicato.

La sintassi è la seguente:

```
        IF [espressione] THEN [n° di linea]
oppure  IF [espressione] GOTO [n° di linea]
oppure  IF [espressione] THEN [altra istruzione BASIC]
```

Se l'espressione presente dopo l'IF è vera, allora si verifica o il salto alla linea indicata dopo il THEN oppure l'esecuzione delle istruzioni presenti dopo il THEN.
L'espressione può riguardare:

— una condizione semplice: ad esempio, A > 0
— una condizione sottintesa: ad esempio, A OR B (si sottintende A OR B = 1, poiché si tratta di un'operazione logica)
— una condizione composta: ad esempio, A = 20 AND B = 13.

Il programma seguente illustra il funzionamento di IF...THEN:

```
 10 INPUT N
 20 IF N > 0 THEN GO TO 100
 30 N = ABS(N)
100 PRINT SQR(N)
110 END
```

oppure:

```
10 INPUT N
20 IF N > 0 THEN N = ABS(N)
30 PRINT SQR(N)
40 END
```

SYS

Questa istruzione permette la chiamata di una routine in linguaggio macchina.
La sintassi è la seguente:

```
SYS(X)
```

X è un indirizzo espresso in numeri decimali, compreso dunque fra 0 e 65535.

Quando, ad esempio, si scrive SYS(5160) si provoca la chiamata della routine in linguaggio macchine che inizia all'indirizzo 5160. Si verifica il ritorno al programma in BASIC quando il microprocessore incontra un'istruzione di ritorno da sottoprogramma (RTS con codice di operazione $60 = 96).

USR

Anche questa istruzione permette la chiamata di una routine in linguaggio macchina, ma questa volta si ha la possibilità di passare un parametro tra la routine e il programma in BASIC.
La sintassi è la seguente:

```
USR(X)
```

Si scriverà ad esempio:

```
Y = USR(X)
```

Prima di chiamare questa routine, bisogna caricare agli indirizzi 785 = $ 311 e 786 = $ 312 il suo indirizzo di partenza (byte più significativo in 785 e byte meno significativo in 786). Il passaggio dei parametri si effettua per mezzo del registro di memorizzazione dei numeri in virgola mobile all'indirizzo 97 = $61, che occupa 5 byte.

2.5.5 Le istruzioni di ingresso/uscita

CLOSE

Questa istruzione provoca la chiusura di un file di dati o di un "canale" di comunicazione aperto tra il Commodore 64 e una periferica.
La sintassi è la seguente:

```
CLOSE n° di file
```

Ad esempio:

```
OPEN 1, 8, 15
CLOSE 1
```

Ritorneremo su questo esempio con l'istruzione OPEN. Va notato che è consigliabile chiudere i "canali" inutilizzati per economizzare sullo spazio di memoria disponibile.

CMD

Questa istruzione permette di utilizzare una periferica in uscita diversa dallo schermo; ad esempio, una stampante, un'unità a disco, un'unità a cassetta, ecc...
La sintassi è la seguente:

```
CMD n° di file [, stringa]
```

Il funzionamento di questo comando è il seguente: il numero del file deve essere già stato definito in via preliminare da un'istruzione OPEN (vedi più avanti). Il comando CMD dirige tutte le informazioni verso la periferica identificata dal numero di file.
La stringa di caratteri (facoltativa) viene inviata verso la periferica in questione (questo può essere utile per stampare un titolo in cima ad un listing di programma).
Per ritornare alla rappresentazione sullo schermo, si utilizza un comando del tipo PRINT #, su cui ritorneremo più tardi.
Ad esempio, per ottenere una stampa su carta si scrive:

```
OPEN 4, 4: CMD 4
```

Se poi battete il comando LIST, il programma in memoria verrà listato sulla stampante. Per ritornare a rivederlo sullo schermo, battete:

```
PRINT# 4: CLOSE 4
```

GET

Questa istruzione permette di introdurre un carattere dalla tastiera senza dover usare il tasto RETURN, durante l'esecuzione di un programma. Questa istruzione viene generalmente inserita in un ciclo di attesa.

Esempio 1:

```
10 GET A$
20 IF A$ = "M" THEN PRINT "BUONGIORNO"
30 IF A$ = "S" THEN PRINT "BUONASERA"
40 GO TO 10
```

Esempio 2:

```
10 FOR I = 1 TO 5
20 GET A
30 IF A = 0 THEN 20
40 NEXT I
50 END
```

Va notato che i caratteri immessi da tastiera sono immagazzinati nel buffer della tastiera stessa. In esso possono essere immagazzinati fino a 10 caratteri, mentre i codici dei tasti immessi dopo il decimo vanno perduti.

GET #

Questa istruzione permette di acquisire dei dati carattere per carattere, come nel caso dell'istruzione GET, ma in questo caso a partire da una periferica qualsiasi.
La sintassi è la seguente:

```
GET # n° di file, lista di variabili
```

Il numero di file indicato corrisponde a un "canale" aperto in precedenza da un'istruzione OPEN. Le variabili possono essere di tipo qualsiasi: intero, reale o stringa.
Se, al momento dell'esecuzione dell'istruzione GET#, non viene ricevuto alcun dato, viene acquisito il valore nullo (per una variabile intera o reale) oppure la stringa vuota (per una variabile di tipo stringa).
Va notato che questa istruzione può essere utilizzata con la periferica n. 3, che altro non è che lo schermo.
In questo caso si ha la lettura di un carattere alla volta e lo spostamento del cursore di uno spazio verso destra.

Esempio:

```
10 OPEN 1, 1, 0
20 OPEN 4, 4
30 GET#1, A$
40 PRINT#4, A$
50 GOTO 30
60 CLOSE 1: CLOSE 4
```

INPUT

Questa istruzione permette di immettere il valore di una variabile a partire dalla tastiera.
La sintassi è la seguente:

```
INPUT ["testo";] variabile 1 [, variabile 2,..., variabile n]
```

Ad esempio:

```
10 INPUT "NUMERO?"; N
20 INPUT "ANIMALE?", A.
30 PRINT A$; N
```

Il funzionamento dell'istruzione INPUT è il seguente:

— se non viene battuto alcun tasto o se non battete il RETURN dopo aver immesso il dato, il Commodore resta in stato di attesa,
— se viene fatto un errore sul tipo della variabile (ad esempio, si immette una stringa di caratteri quando la variabile dichiarata è di tipo reale), il BASIC invia il messaggio REDO FROM START e domanda di immettere di nuovo la variabile con un punto interrogativo,
— per immettere più dati, bisogna utilizzare una virgola per separare ciascuno di essi,
— se non sono stati introdotti tutti i dati necessari, vedrete apparire sullo schermo un doppio punto interrogativo ("??").

Va notato che INPUT può essere utilizzata solo nel corso di un programma.

INPUT #

Questa istruzione permette di immettere valori di variabili a partire da una periferica qualsiasi.
La sintassi è la seguente:

INPUT # n° di file, variabile 1 [, variabile 2, ..., variabile n]

L'istruzione è utilizzata soprattutto per leggere dati su una cassetta o su un disco, ma può essere usata anche per letture a partire dalla tastiera; tuttavia in questo caso non verrà visualizzato il punto interrogativo come nel caso dell'istruzione INPUT.
Esempio di lettura da una cassetta:

```
10 OPEN 4, 1, 0
20 INPUT#4, A, B, C$, D%
30 CLOSE 4
```

Ogni valore di variabile memorizzato su cassetta o disco deve essere seguito da un RETURN (CHR$(13)).

OPEN

Questa istruzione permette di aprire un canale destinato a veicolare le informazioni tra il Commodore 64 e una periferica.
La sintassi è la seguente:

OPEN n° di file, [n° di periferica] [, comando] [, "nome di file"] [, tipo] [, "modo"]

Il funzionamento di questa istruzione è il seguente:

— il numero di file è un numero compreso fra 1 e 255 ed è utilizzato insieme con le istruzioni CLOSE, CMD, GET #, INPUT #, PRINT #. Questo numero determina un canale al quale è associato un numero di periferica
— il numero di periferica è stabilito per ciascuna di esse e non può essere modificato. Diamo il numero caratteristico di ciascuna periferica:

- Tastiera: 0
- Lettore di cassette: 1
- Modem, RS-232: 2
- Schermo: 3
- Stampante: 4 o 5
- Unità a disco: da 8 a 11

— il comando permette di trasmettere ordini specifici a certe periferiche ed è perciò caratteristico di ciascuna di esse. Anche in questo caso diamo il significato di alcuni di essi:

- lettore di cassette: 0 → lettura
  1 → scrittura
  2 → scrittura con indicatore di fine nastro dopo il programma

- stampante: 0 Maiuscolo/Grafico
  7 Maiuscolo/Minuscolo

- unità a disco: 2—14 permette di aprire un canale per il trasferimento di dati
  15 permette di inviare un comando al disco (ad esempio, formattazione).

Se il numero della periferica viene omesso, il Commodore 64 ritiene che i trasferimenti avranno luogo con il lettore di cassette. Inoltre se il comando non viene specificato (sempre nel caso di un trasferimento di dati con il registratore a cassetta), il calcolatore ritiene che il valore sia 0 (lettura)

— il nome del file è caratterizzato da una stringa composta da un massimo di 16 caratteri. E' facoltativo nel caso di trasferimenti con lettore di cassette o stampante. Invece è obbligatorio nel caso di utilizzo di un'unità a disco
— le informazioni supplementari di "tipo" e "modo" sono utilizzate unicamente con l'unità a disco. Esse permettono di definire dei file di tipo diretto o sequenziale, in modalità di lettura o scrittura. Se il "tipo" e il "modo" sono omessi, il calcolatore ritiene che il file è di tipo programma.

Non entreremo in ulteriori dettagli sui diversi tipi di file, perché questo esula dai compiti di questo libro. Chi è interessato potrà consultare il manuale fornito con l'unità a disco modello 1541.

Ecco alcuni esempi sull'uso di OPEN:

```
OPEN 5, 0                 ... lettura da tastiera
OPEN 5, 1, 0, "NOME"...   lettura da cassetta
OPEN 5, 1, 1, "NOME"...   scrittura su cassetta
OPEN 5, 1, 2 "NOME" ...   scrittura su cassetta con machio
                          di fine nastro
OPEN 10, 3                ... scrittura su schermo
OPEN 20, 4                ... stampa su stampante tipo 4
OPEN 20, 5, 7, "NOME" ... stampa su stampante tipo 5
                          in Maiuscolo/Minuscolo
OPEN 1, 8, 15,
"COMANDO"                 ... invio di comando al disco
```

PRINT

Questa istruzione provoca la visualizzazione sullo schermo dei testi e dei valori delle variabili precisati nell'istruzione stessa.

La sintassi è la seguente:

```
PRINT variabile 1 [, variabile 2,..., variabile n]
```

La sintassi che abbiamo fornito permette di aggiungere anche testi che però vanno messi tra virgolette. La virgola di separazione può essere sostituita da un punto-e-virgola, ma il formato di stampa sarà differente.
Lo schermo visualizza 40 caratteri per linea, ma il comando di PRINT permette di visualizzare fino a 80 caratteri (cioè due linee). Il comando è suddiviso in 8 zone di 10 caratteri ciascuna. Una virgola tra due variabili o due frasi provoca la visualizzazione della seconda di esse all'inizio della zona seguente. Al contrario un punto-e-virgola provoca la visualizzazione di questi due elementi uno dopo l'altro (con uno spazio vuoto di separazione nel caso di variabili numeriche).

Ecco alcuni esempi sull'uso di PRINT:

```
PRINT 5, 10, 15                dará:
5         10         15

PRINT 5; 10; 15                dará:
5 10 15

PRINT "PARIS"; "LE MANS"       dará:
PARISLEMANS    (tutto unito)

PRINT "PARIS", "LE MANS"       dará:
PARIS          LE MANS
```

L'istruzione PRINT può anche essere utilizzata nel caso di un programma per effettuare spostamenti di cursore, stampa con caratteri su sfondo invertito, cambiamenti di colore, ecc. (come vedremo più avanti). Basta cominciare l'elenco di questi caratteri speciali con le virgolette (tasti SHIFT e 2)

1) Movimento del cursore
I tasti che possono essere programmati sono i seguenti:

CLR/HOME
SHIFT CLR/HOME
⇑ CRSR ⇓
SHIFT ⇑ CRSR ⇓
⇐ CRSR ⇒
SHIFT ⇐ CRSR ⇒

Questi tasti possono essere utilizzati in sequenza per spostare le scritte nella posizione voluta sullo schermo.

2) Caratteri invertiti

Per visualizzare caratteri in "video invertito" basta battere i tasti CTRL e 9. Per ritornare nella modalità normale di visualizzazione

basta battere CTRL 0 o RETURN.

3) Colore dei caratteri
Il colore dei caratteri visualizzati sul video può essere modificato utilizzando le seguenti combinazioni di tasti:

CTRL 1 nero
CTRL 2 bianco
CTRL 3 rosso
CTRL 4 celeste
CTRL 5 porpora
CTRL 6 verde
CTRL 7 blu
CTRL 8 giallo

C= 1	(C= è il tasto col logotipo Commodore)	arancio
C= 2		marrone
C= 3		rosso chiaro
C= 4		grigio 1
C= 5		grigio 2
C= 6		verde chiaro
C= 7		azzurro
C= 8		grigio 3

PRINT #

Questa istruzione permette di inviare valori delle variabili o testi alla periferica col numero di file precisato.

La sintassi è la seguente:

PRINT = n° di file [, variabile 1] [, variabile 2] ... [, variabile n]

Ad esempio per una rappresentazione su stampante si scriverà:

```
10 PRINT  "SU SCHERMO"
20 OPEN 1, 4
30 PRINT#1,"SU STAMPANTE"
40 CLOSE 1
```

I segni di punteggiatura tra variabili e frasi (virgole e punti-e-virgola) operano come nel caso dell'istruzione PRINT.

2.5.6 L'Editing dello schermo

POS

Questa istruzione fornisce la posizione del cursore sullo schermo.

La sintassi è la seguente:

```
POS(0)
```

Il valore ottenuto è compreso tra 0 e 79. Il Commodore 64 possiede una visualizzazione "reale" di 40 caratteri per linea e una visualizzazione "logica" di 80 caratteri; è evidente dunque che ogni valore compreso fra 40 e 79 farà riferimento alla seconda linea sullo schermo. Ad esempio:

```
IF POS(0) > = 39 THEN PRINT CHR$(13)
```

SPC

Questa istruzione è utilizzata per formattare una rappresentazione sul video o su una qualsiasi altra periferica.

La sintassi è la seguente:

SPC(X) con $0 \leq X \leq 255$

per lo schermo e il lettore di cassette (con $0 \leq X \leq 254$ per l'unità a disco).
Questa istruzione provoca la visualizzazione di X spazi vuoti sullo schermo o sulla periferica, a partire dalla prima posizione disponibile.

TAB

Questa istruzione permette il posizionamento del cursore alla posizione X dello schermo.

La sintassi è la seguente:

TAB(X) con $0 \leqslant X \leqslant 255$

L'istruzione TAB deve essere utilizzata unicamente con l'istruzione PRINT. Non funziona con l'istruzione PRINT #.

Ad esempio:

```
10 INPUT N
20 PRINT "NUMERO" TAB(10); N
```

2.5.7 Le altre istruzioni

CLR

Questa istruzione inizializza di nuovo tutte le variabili, le tabelle e le stringhe così come il puntatore DATA (analogamente a quanto fa l'istruzione RESTORE). Ritorneremo su queste nozioni nel prossimo paragrafo.
Il risultato per l'utilizzatore è che tutte le variabili, tutte le tabelle sono azzerate e le stringhe sono tutte annullate.

DATA

Questa istruzione permette l'immagazzinamento dei dati all'interno di un programma in BASIC.
La sintassi è la seguente:

DATA valore 1 [, valore 2, ..., valore n]

Essa può essere collocata in qualsiasi punto del programma e viene utilizzata congiuntamente con l'istruzione READ.
In una linea DATA si possono utilizzare dati relativi a qualsiasi tipo di variabile (intera, reale, stringa), separandoli tra loro per mezzo di una virgola.
Va notato che il Commodore 64 utilizza un puntatore DATA che viene incrementato ogni volta che viene eseguita un'istruzione READ.
Quando incontreremo l'istruzione READ daremo un esempio applicativo.

DEF FN

Questa istruzione permette all'utente di programmare una sua funzione.
La sintassi è la seguente:

```
DEF FN [nome] (variabile) = espressione matematica
```

Il nome può comprendere uno o due caratteri alfanumerici, di cui il primo deve obbligatoriamente essere una lettera.

Esempio:

```
DEF FNA1(X) = 2*π*X
```

Nel seguito questa funzione può essere chiamata come ogni altra variabile grazie all'istruzione FN. Così C = FNA1(4) darà come risultato C = 25.13.

DIM

Si tratta di un'istruzione per la dichiarazione del formato delle tabelle.
La sintassi è la seguente:

```
DIM variabile 1 (dim 1, dim 2, ..., dim n) [, variabile 2 (...), ..., variabile n
(...)]
```

Le variabili possono essere di tipo intero (%), reale o stringa ($). A una tabella non dichiarata da un'istruzione DIM è automaticamente attribuito un valore 10 per ciascuna dimensione:
Va notato che una tabella non può essere dimensionata che una volta nel corso di un programma. La dimensione massima (dim i) è 32767.
Esempio: una tabella dichiatata da DIM A (2,1) contiene i 6 elementi seguenti:

A(0,0)
A(1,0)
A(2,0)
A(0,1)
A(1,1)
A(2,1)

FN

Questa istruzione si riferisce a una funzione utente definita in precedenza da un DEF FN.
La sintassi è la seguente:

FN [nome] (numero)

Esempio:

PRINT FNA (10)

FOR...TO...STEP

Si tratta del classico ciclo FOR...NEXT.
La sintassi è la seguente:

FOR variabile = inizio TO fine [STEP passo]

Un contatore di ciclo viene inizializzato al valore "inizio" e viene incrementato ad ogni ciclo eseguito del valore indicato da "passo" fino al raggiungimento del valore "fine". Se non viene precisato alcun passo, l'incremento è uguale a 1.
Va notato che un ciclo FOR...NEXT viene sempre eseguito almeno una volta, qualunque siano i valori di "inizio" e "fine".

Esempio:

```
10 FOR I = 1 TO 10
20 PRINT I*4
30 NEXT I
```

FRE

Questa istruzione fornisce il numero di byte che sono ancora disponibili nella RAM per la memorizzazione del programma in BASIC.

La sintassi è la seguente:

FRE(X)

Il numero tra parentesi può assumere qualsiasi valore.
Si potrà dunque scrivere:

```
PRINT FRE(0)
```

Il risultato è normalmente compreso fra 0 e 38911. Se il risultato è negativo basta sommare 65536 al risultato ottenuto per conoscere la qualità di memoria RAM disponibile.

LET

Questa istruzione viene utilizzata per attribuire un certo valore a una variabile.
La sintassi è la seguente:

```
LET variabile = espressione
```

Si scriverà ad esempio LET A = 10. In realtà questa istruzione è piuttosto inutile e non serve che a consumare spazio di memoria. Basta scrivere semplicemente: A = 10.

NEXT

Questa istruzione indica la fine di un ciclo FOR...NEXT.

PEEK

Questa istruzione va a leggere il contenuto della locazione di memoria di indirizzo X.
La sintassi è dunque la seguente:

PEEK(X) $0 \leqslant X \leqslant 65535$

Esempio:

```
Y = PEEK(X)
oppure:
PRINT PEEK(X)
```

Il risultato è dato in forma decimale ed è perciò compreso fra 0 e 255.

POKE

Questa istruzione permette di attribuire un valore in una certa locazione di memoria.
La sintassi è la seguente:

POKE indirizzo, valore

con: $0 \leqslant$ indirizzo $\leqslant 65535$

$0 \leqslant$ valore $\leqslant 255$

Esempio:

```
POKE 785,00
```

READ

Questa istruzione permette di assegnare dei valori ad alcune variabili nel corso dell'esecuzione di un programma. Va utilizzata congiuntamente con l'istruzione DATA.
La sintassi è la seguente:

READ variabile 1 [, variabile 2, ..., variabile n]

Esempio:

```
10 DATA 10, 20, 30, "MUCCA"
20 READ A, B, C, D$
```

Ad A sarà dunque attribuito il valore 10, a B il valore 20, a C il valore 30 e alla stringa D$ la parola MUCCA.
Una medesima variabile può accedere a dati diversi e a differenti linee DATA.

Esempio:

```
10 DATA 1, 2, 3, 4, 5
20 DATA 10, 20, 30, 40, 50
30 FOR I = 1 TO 10
40 READ A
50 PRINT A
60 NEXT I
```

Infatti ad ogni istruzione READ, viene incrementato un puntatore che va a posizionarsi sul dato successivo. Questo puntatore può essere nuovamente inizializzato per mezzo dell'istruzione RESTORE come verrà precisato più avanti.
Va notato che il numero di READ fra due RESTORE deve essere sempre inferiore o uguale al numero di elementi presenti nell'insieme delle linee DATA del programma.

REM

L'istruzione REM indica una linea di commento, utile per la "documentazione" del programma.

Esempio:

```
10 REM PROGRAMMA DI GIOCO
```

Al momento dell'esecuzione del programma, l'interprete salta la linea che contiene un REM e passa a quella successiva.

RESTORE

Questa istruzione, usata congiuntamente con le istruzioni DATA e READ, permette di azzerare il puntatore di DATA, che va a posizionarsi sul primo elemento della linea DATA presente nel programma.
Il programma che segue illustra il funzionamento delle istruzioni DATA, READ, RESTORE e permette di conoscere i giorni della settimana in funzione della data.

```
10 DEF FNA(X) = X-4*INT(X/4)
20 DEF FNB(X) = X-7*INT(X/7)
30 INPUT G, M, A
40 PRINT "IL"; G; ""; M; ""; A; " E' UN"
50 IF M< =2 THEN 70
60 N = 0: GO TO 110
70 IF FNA(A) = 0 THEN 90
80 N = 2: GO TO 110
90 IF A = 0 THEN 80
100 N = 1
110 C = INT(365.25*A) + INT(30.56*M) + G + N
120 I = FNB(C)
130 FOR J = 1 TO I + 1: READ A$
140 NEXT J
150 PRINT A$
160 RESTORE
170 DATA "MERCOLEDI", "GIOVEDI", "VENERDI", "SABATO"
    DATA "DOMENICA", LUNEDI", "MARTEDI"
180 END
```

Dopo aver introdotto il programma nel calcolatore, battete RUN e introducete il giorno, il mese e l'anno (< = 99)

STATUS

Questa istruzione permette di conoscere il valore del registro di stato (STATUS) dopo un'operazione di ingresso/uscita (su cassetta, disco, schermo, tastiera...).
Per conoscere con precisione i valori che possono essere assunti da questo registro, vi consigliamo di consultare il manuale del Commodore 64.

TI

Questa istruzione permette di leggere l'orologio in tempo reale presente sul Commodore 64. Questo orologio viene azzeratò quanto si accende il calcolatore e poi aggiornato ogni 60.mo di secondo. Battendo PRINT TI otterrete il numero di sessantesimi di secondo che sono trascorsi dopo aver acceso il calcolatore.

TI$

Questa istruzione permette di accedere all'orologio in tempo reale, espresso però questa volta in ore, minuti e secondi.
Per far partire l'orologio si scrive:

```
TI$ = "HHMMSS"
```

Esempio: `TI$ = "122834"`

cioè ore 12, 28 minuti e 34 secondi.
Per leggere l'orologio si scrive:

esempio:

```
PRINT TI$
```

WAIT

Questa istruzione non sarà mai utilizzata da quanti utilizzano il Commodore 64 così com'è. Invece sarà molto utile a quanti vogliono utilizzare la loro macchina come strumento di controllo e realizzare funzioni di ingresso/uscita con il mondo esterno.

La sintassi è la seguente:

```
WAIT indirizzo, maschera 1 [, maschera 2]
```

Questa istruzione merita una breve spiegazione. L'interprete BASIC effettua un "E logico" (AND) fra il contenuto dell'indirizzo specificato e la "maschera" n. 1.
Inoltre se è presente una maschera n. 2, esso effettua un "O esclusivo" tra il risultato ottenuto e la seconda maschera. In altre parole, l' "E logico" permette di conservare soltanto i "bit utili" (quelli che si desidera provare).
Si ha: X AND 0 = 0 e X AND 1 = 1. La maschera n. 1 genererà (in binario) degli "uno" per ciascun bit che si desidera provare e degli "zero" per i bit "inutili". Successivamente la maschera n. 2 permette di provare il valore dei bit che rimangono, cioè i bit "utili".

Si ha: $X \veebar 1 = \overline{X}$ e $X \veebar 0 = X$.

2.6 I puntatori del Commodore 64

Raggruppiamo sotto la denominazione di puntatore l'insieme o gli insiemi delle locazioni di memoria utilizzate dall'interprete BASIC e necessarie al suo funzionamento.
Questi puntatori vengono inizializzati in modo automatico quando accendiamo il Commodore 64; tuttavia possono essere modificati dall'utilizzatore.
Una buona conoscenza di questi puntatori e del loro ruolo ci sembra assolutamente necessario alla comprensione del funzionamento logico del Commodore 64.
In questo paragrafo descriveremo il ruolo dei puntatori più utili che hanno il compito di gestire lo spazio di memoria disponibile per la conservazione dei programmi in BASIC.

1) Puntatore di inizio BASIC

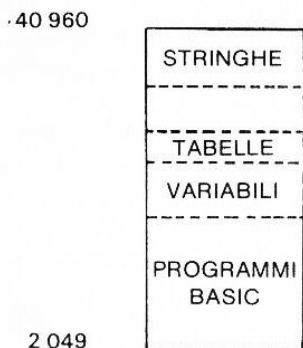
Il Commodore 64 possiede una certa quantità di RAM destinata alla memorizzazione dei programmi in BASIC e di certi dati necessari al funzionamento dell'interprete. Fra questi esiste un puntatore il cui ruolo è di precisare l'indirizzo della prima locazione di memoria disponibile per la memorizzazione di un programma BASIC. Per essere conosciuto questo indirizzo necessita di due locazioni di memoria situate nella pagina "zero" agli indirizzi 43 = $2B e 44 = $2C.
In 43 si trova il byte meno significativo e in 44 quello più significativo.
Per ottenere l'indirizzo iniziale di un programma in BASIC basta dunque battere:

```
PRINT PEEK(43) + 256*PEEK(44)
```

che fornisce normalmente l'indirizzo 2049.

2) Puntatore di inizio della zona di memorizzazione delle variabili

Lo spazio RAM disponibile serve non solo a conservare il programma BASIC, ma anche le diverse variabili, le tabelle e le stringhe.

La RAM del Commodore 64 è strutturata secondo il seguente schema:

Quando, nel corso dell'esecuzione di un programma BASIC, l'interprete incontra una variabile, la immagazzina nella zona riservata a questo scopo e situata proprio alla fine del programma.
Analogamente quando l'interprete incontra una tabella la immagazzina subito dopo la zona riservata alla conservazione delle variabili. Questa zona varia naturalmente in funzione delle dimensioni del programma e del numero delle variabili utilizzate.
Invece, quando l'interprete BASIC calcola una nuova stringa di caratteri, la immagazzina nella zona riservata a questo scopo e situata nella parte più alta della memoria disponibile, a cominciare dalla fine (indirizzo 40960). Il puntatore di inizio delle variabili indica l'inizio della zona in cui esse verranno immesse. Esso è situato agli indirizzi 45 = $2D e 46=$2E. Per conoscere l'inizio di questa zona di memoria basterà dunque battere:

```
PRINT PEEK(45) + 256*PEEK(46)
```

che fornisce l'indirizzo 2051, quando non è presente alcun programma in memoria.
Va notato che questo puntatore indica anche la fine di un programma in BASIC.
Esaminiamo ora più in dettaglio come le diverse variabili sono immagazzinate in memoria.
Abbiamo già visto che sul Commodore 64 esistono tre tipi di variabili: le variabili reali, le variabili intere e le variabili di stringa.

Variabili di tipo intero

Una variabile intera è caratterizzata da:

— nome (due caratteri alfanumerici)
— valore (codificato su 2 byte).

Consideriamo la seguente istruzione BASIC:

```
10 A% = 20
```

Battete questa linea sulla tastiera, ordinate un RUN e poi battete:

```
PRINT PEEK(45) + 256*PEEK(46)
```

che fornisce il valore 2061.
Battete in seguito:

```
PRINT PEEK(2061)     che dà 193
PRINT PEEK(2062)     che dà 128
PRINT PEEK(2063)     che dà 0
PRINT PEEK(2064)     che dà 20
PRINT PEEK(2065)     che dà 0
PRINT PEEK(2066)     che dà 0
PRINT PEEK(2067)     che dà 0
PRINT PEEK(2068)     che dà un valore qualunque.
```

All'indirizzo 2061 si ha 193 = 128 + 65 = 128 + ASC ("A").
All'indirizzo 2062 si ha 128 = 128 + 0 = 128 + ASC(" ").

La codifica è dunque molto semplice:

— sono necessari 7 byte
— i primi due byte sono utilizzati per il nome della variabile: il codice presente in memoria è ottenuto aggiungendo al codice ASCII di ciascun carattere che rappresenta il nome della variabile il valore 128 che è destinato ad indicare all'interprete BASIC che si trova in

presenza di una variabile di tipo intero e che basteranno solo due byte per conoscerne il valore
— i due byte successivi rappresentano il valore della variabile intera, la cui codifica è stata fornita in dettaglio all'inizio di questo capitolo (rappresentazione in complemento a 2 su 16 bit). Va notato solo che il byte di peso più forte (MSB) è immagazzinato per primo
— i tre byte successivi hanno valore "zero". Non sono di alcuna utilità nel caso di una variabile intera. Servono unicamente a conservare una certa omogeneità nella memorizzazione delle variabili (che utilizza 7 byte).

Variabili di tipo reale
Una variabile di tipo reale è caratterizzata da:

— nome (due caratteri)
— valore (su 5 byte)

Ad esempio, battete:

```
10 B1 = -38.56 E5
```

battete RUN e poi:

```
PRINT PEEK(45) + 256*PEEK(46)
```

che dà il valore 2067.

Eseguite poi:

```
PRINT PEEK(2067)        che dà  66
PRINT PEEK(2068)        che dà  49
PRINT PEEK(2069)        che dà 150
PRINT PEEK(2070)        che dà 235
PRINT PEEK(2071)        che dà  90
PRINT PEEK(2072)        che dà   0
PRINT PEEK(2073)        che dà   0
```

All'indirizzo 2067 si ha 66 = ASC("B").
All'indirizzo 2068 si ha 49 = ASC("1").
La codifica necessita di 7 byte:

— i primi due byte sono utilizzati per il nome della variabile: in questo caso il codice presente in memoria è il codice ASCII dei caratteri che rappresentano il nome della variabile. Il valore "128" non è stato aggiunto per permettere all'interprete BASIC di capire che si tratta di una variabile di tipo reale
— i cinque byte successivi rappresentano il valore della variabile reale in virgola mobile descritta all'inizio di questo capitolo.

Variabili di tipo stringa

Prima di descrivere le modalità con cui sono memorizzate le stringhe di caratteri, occorre fare una precisazione: quando l'interprete BASIC incontra per la prima volta una stringa (ad esempio A$ = "TAVOLO"), l'interpreta come una variabile e la cataloga in memoria secondo una codifica che spiegheremo più avanti. Vedremo in particolare che per conoscere una stringa, basta conoscere l'indirizzo dell'inizio della sua memorizzazione e la sua lunghezza. Qui viene conosciuta la stringa A$ = "TAVOLO" e immessa nel programma BASIC.
Se poi si considera la stringa B$=LEFT$(A$,3), è necessario memorizzare i tre caratteri "T", "A", "V" che appartengono a questa stringa. B$ sarà sempre considerata una variabile di tipo stringa, ma la sua conservazione in memoria non avverrà in mezzo al programma BASIC ma nella zona di memorizzazione delle stringhe (nella parte alta della RAM disponibile).
Esaminiamo ora più da vicino le modalità che presiedono alla memorizzazione delle stringhe. Esse sono caratterizzate da:

— nome (due caratteri alfanumerici)
— lunghezza (numero dei caratteri che formano la stringa: meno di 255)
— indirizzo di inizio della memorizzazione in RAM.

Consideriamo ad esempio la linea BASIC:

```
10 C6$ = "COMMODORE-64"
```

Dopo aver battuto l'istruzione, battete RUN e:

```
PRINT PEEK(45) + 256*PEEK(46)
```

che fornirà il valore 2074.
Battete poi:

```
PRINT PEEK(2074)     che dà 67
PRINT PEEK(2075)     che dà 182
PRINT PEEK(2076)     che dà 12
PRINT PEEK(2077)     che dà 10
PRINT PEEK(2078)     che dà 8
PRINT PEEK(2079)     che dà 0
PRINT PEEK(2080)     che dà 0
```

All'indirizzo 2074 si ha 67 = ASC("C").
All'indirizzo 2075 si ha 182 = 128 + ASC("6").

La codifica avviene così:

— i primi due byte, come nel caso delle variabili intere e reali, sono utilizzate per il nome della variabile. Il primo byte è il codice ASCII del primo carattere che rappresenta il nome della variabile. Il secondo byte è ottenuto aggiungendo il valore 128 al codice ASCII del secondo carattere del nome. Questa codifica serve a indicare all'interprete BASIC che si trova di fronte a una variabile di stringa
— il terzo byte dà la lunghezza della stringa
— il quarto e il quinto byte danno l'indirizzo di inizio della stringa nella memoria. In questo caso essa si trova all'interno del programma BASIC.

Per trovare l'indirizzo della catena C6$ basta battere:

```
PRINT PEEK(2077) + 256*PEEK(2078)
```

3) Puntatore di inizio di tabella

Come abbiamo già detto, il Commodore 64 memorizza le tabelle subito dopo le variabili in una zona il cui inizio è dato da un puntatore

situato agli indirizzi 47 = $2F e 48 = $30. Va notato che ogni volta che una nuova variabile viene introdotta nel corso del programma, tutta questa zona di memorizzazione delle tabelle deve essere spostata, è dunque nostro interesse utilizzarla il meno possibile (soprattutto non sovradimensionarla) per non rallentare lo svolgimento del programma.
Esaminiamo dunque le modalità di memorizzazione delle tabelle. Una tabella è completamente determinata se conosciamo:

— nome (due caratteri alfanumerici)
— tipo (intero, reale, stringa); va notato che, come nel caso delle variabili, il tipo della tabella è indicato nei due byte riservati al nome aggiungendo, o meno, il valore 128
— dimensione (1 byte)
— numero di elementi per dimensione (2 byte per dimensione)
— valore di ogni elemento: codifica su 5 byte per i reali, 2 byte per gli interi, 3 byte (lunghezza + indirizzo di memorizzazione) per le stringhe.

Consideriamo l'esempio del programma seguente:

```
10 DIM A1%(1,1)
20 A1%(0,0) = 2
30 A1%(1,0) = 3
40 A1%(0,1) = 4
50 A1%(1,1) = 5
60 DIM A1%(0,1)
70 A2$(0,0) = "COMMODORE"
80 A2$(0,1) = "64"
90 END
```

Immettere il programma nel calcolatore, battete RUN e poi:

```
PRINT PEEK(47) + 256*PEEK(48)
```

che vi darà il valore 2190. In memoria si avrà:

```
PEEK(2190) = 193 = 128 + ASC("A")  } nome A1, tipo intero
PEEK(2191) = 177 = 128 + ASC("1")  }

PEEK(2192) =  17                   } byte usati per memorizzare
PEEK(2193) =   0                   } la tabella

PEEK(2194) =   2                     dimensioni

PEEK(2195) =   0                   } numero di elementi
PEEK(2196) =   2                   } della dimensione destra

PEEK(2197) =   0                   } numero di elementi
PEEK(2198) =   2                   } della dimensione sinistra

PEEK(2199) =   0                   } elemento A1% (0,0)
PEEK(2200) =   2                   }

PEEK(2201) =   0                   } elemento A1% (1,0)
PEEK(2202) =   3                   }

PEEK(2203) =   0                   } elemento A1%(0,1)
PEEK(2204) =   4                   }

PEEK(2205) =   0                   } elemento A1%(1,1)
PEEK(2206) =   5                   }

PEEK(2207) =  65 = ASC("A")        } nome A2, tipo stringa
PEEK(2208) = 178 = ASC("2")+ 128   }
PEEK(2209) =  15                   } byte necessari per
PEEK(2210) =   0                   } memorizzare la tabella
PEEK(2211) =   2                     dimensioni
PEEK(2212) =   0                   } numero di elementi
PEEK(2213) =   2                   } della dim. destra
PEEK(2214) =   0                   } numero di elementi
PEEK(2215) =   1                   } della dim. sinistra
PEEK(2216) =   9                     A2$(0,0): numero di caratteri
PEEK(2217) = 105                   } A2$(0,0): indirizzo di memoria
PEEK(2218) =   8                   }

PEEK(2219) =   2                     A2$(0,1): numero di caratteri
PEEK(2220) = 130                   } A2$(0,1): indirizzo di memoria
PEEK(2221) =   8                   }
```

4) Puntatore di fine tabella

Questo puntatore, come indica il nome, segna la fine della zona di memorizzazione delle tabelle. E' situato agli indirizzi 49 = $31 e 50 =

$32. Dopo aver introdotto il programma precedente e battuto RUN, potete fare:

```
PRINT PEEK(49) + 256 * PEEK(40)
```

e otterrete il valore 2592.

5) Puntatore della zona delle stringhe

Abbiamo visto che le stringhe di caratteri calcolate al momento dell'esecuzione di un programma sono immagazzinate in una zona riservata situata nella parte alta della memoria RAM disponibile. Esiste dunque un puntatore, collocato agli indirizzi 51 = $33 e 52 = $34, che parte dall'alto di questa memoria e si sposta verso il basso ogni volta che viene calcolata una nuova stringa. Infatti in questa zona sono conservati semplicemente i codici ASCII dei caratteri che compongono le stringhe, mentre queste ultime, come abbiamo visto, sono catalogate nella zona riservata alla memorizzazione delle variabili.

Esempio:

```
10 C6$ = "COMMODORE-64"
20 B$ = LEFT$(A$,3)
30 C$ = RIGHT$(A$,2)
```

Fate RUN poi:

```
PRINT PEEK(51) +256 * PEEK(52)
```

che darà il valore 40955. In memoria si ottiene allora:

```
PEEK(40955) = 54 = ASC("6")
PEEK(40956) = 52 = ASC("4")
PEEK(40957) = 67 = ASC("C")
PEEK(40958) = 79 = ASC("O")
PEEK(40959) = 77 = ASC("M")
```

6) Puntatore di fine memoria

La parte alta della RAM riservata alla conservazione dei programmi in BASIC è data da un puntatore collocato agli indirizzi 55 = $37 e 56 = $38.

Battendo:

```
PRINT PEEK(55) + 256 * PEEK(56)
```

ottenete il valore 40960.

Modificando il contenuto delle locazioni di memoria di indirizzi 51, 52 (puntatore della zona delle stringhe) e 55, 56 (puntatore di fine memoria) è possibile proteggere una parte della memoria RAM dalla accessibilità da parte del BASIC; questo può essere utile quando si vogliono immagazzinare dei programmi in linguaggio macchina o programmarsi un proprio generatore di caratteri, come vedremo più avanti.

7) Puntatore del numero della linea DATA

Questo puntatore, situato agli indirizzi 63 = $3F e 64 = $40, indica, al momento di una READ, il numero della linea in cui si effettua la lettura, cioè il numero della linea in cui si trova l'istruzione DATA.

8) Puntatore dell'indirizzo di DATA

Al momento di una READ, l'interprete BASIC percorre la lista dei dati contenuti nell'istruzione DATA, da sinistra verso destra. E' dunque necessario disporre di un puntatore che indichi l'indirizzo del dato successivo. Questo puntatore è situato agli indirizzi 65 = $ 41 e 66 = $42 e indica l'indirizzo della virgola che separa il dato appena letto da quello da leggere.

Il seguente programma illustra il suo funzionamento:

```
 10 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
 20 FOR I = 1 TO 5
 30 READ A
 40 PRINT A: NEXT I
 50 POKE 785, PEEK(65)
 60 POKE 786, PEEK(66)
 70 RESTORE
 80 FOR I = 1 TO 9
 90 READ A
100 PRINT A: NEXT I
110 POKE 65, PEEK(785)
120 POKE 66, PEEK(786)
130 FOR I = 1 TO 4
140 READ A
150 PRINT A: NEXT I
160 END
```

Gli indirizzi 785 e 786 sono utilizzati per conservare temporaneamente il contenuto del puntatore dell'indirizzo di DATA. Probabilmente avrete capito che si tratta del vettore utilizzato dall'istruzione USR. Detto questo, abbiamo finito con il BASIC del Commodore 64. A questo punto possedete le basi necessarie per affrontare la parte dedicata alla visualizzazione con lo studio del controllore dello schermo video.

3
Il controllore video

3.1 Introduzione

Il controllore video è un circuito specializzato che ha il compito di generare tutti i segnali necessari a comandare un monitor o un televisore a colori in maniera totalmente trasparente per l'utilizzatore. Questi non deve per nulla preoccuparsi della sua programmazione per mezzo dei 47 registri di controllo specializzati. Le informazioni visualizzate sullo schermo (caratteri alfanumerici e semigrafici, grafici ad alta risoluzione, SPRITE) provengono dalla memoria del Commodore 64 alla quale il controllore video può accedere senza interferire sul funzionamento del microprocessore.
I registri di controllo sono situati tra gli indirizzi 53248 (= $D000) e 53294 (= $D02E) e permettono funzioni molto diverse:

— rappresentazione alfanumerica standard (25 righe di 40 caratteri)
— rappresentazione alfanumerica a più colori
— rappresentazione grafica ad alta risoluzione (300 x 200 punti)
— SPRITE (che significa folletto).

Nelle pagine che seguiranno descriveremo il modo di programmare i diversi registri per sfruttare a fondo le possibilità di questo circuito integrato.

3.2 Modo standard di visualizzazione

Nella modalità standard (quella che selezioniamo ogni volta che accendiamo il Commodore 64) la visualizzazione avviene, come ab-

biamo detto, su 25 righe di 40 caratteri. I caratteri visualizzati si trovano in una parte della RAM chiamata memoria-schermo. Ciascuno di essi necessita di un byte; dunque la rappresentazione sullo schermo in modalità alfanumerica utilizza 1000 locazioni di memoria consecutive.
Normalmente questa zona di memoria è collocata fra gli indirizzi 1024(=$0400) e 2023(=$07E7).
Ciò nonostante questa posizione può essere facilmente modificata per mezzo del registro di indirizzo 53272 (=$D018) presente sul controllore video.
Ma, prima di descrivere la maniera di procedere, dobbiamo entrare un po' in dettaglio nelle modalità di funzionamento di questo circuito. Quest'ultimo è stato progettato per indirizzare fino a 16 kbyte di memoria (ad esempio, nel caso di una rappresentazione grafica ad alta risoluzione). Poiché il Commodore 64 possiede (come indica il suo stesso nome) 64 kbyte di memoria RAM, può essere utile la possibilità di accedere alla totalità di questi 64 k, invece che 16 k; questo può avvenire, ad esempio, quando si programmano dei giochi con animazioni.
Questo è reso possibile dalla "paginazione" della memoria RAM, cioè dalla sua divisione in 4 blocchi di 16 kbyte. Ciascuno di questi blocchi potrà essere selezionato per mezzo dei bit 0 e 1 della porta A del circuito di controllo ingresso/uscita n. 2 tipo 6526. Una descrizione dettagliata di questo circuito esula dagli scopi del nostro libro.

Per poter selezionare una delle quattro "pagine" di memoria, i bit 0 e 1 di questa porta devono essere programmati in uscita.
La procedura da seguire è la seguente.

Battere:

```
POKE 56578, PEEK(56578) OR 3
```

Questo permette di configurare in uscita i bit 0 e 1 del registro DDRA (Data Direction Register A). Battete ora:

```
POKE 56576, (PEEK(56576) AND 252) OR X
```

dove X è un intero che può prendere i valori 0, 1, 2 o 3.

La tavola riportata qui sotto fornisce la posizione dello spazio indirizzabile del controllore video in funzione del valore di X.

X	N° di pagina	Spazio indirizzabile
0	3	49 152 – 65 535 : $C000 – $FFFF
1	2	32 768 – 49 151 : $8000 – $BFFF
2	1	16 384 – 32 767 : $4000 – $7FFF
3	0	0 – 16 383 : $0000 – $3FFF

Questo concetto di paginazione è importantissimo, come spiegheremo più avanti. Va notato che al momento dell'accensione del calcolatore viene selezionata la pagina n. 0.
Torniamo ora alla rappresentazione alfanumerica

1) La memoria - schermo

Abbiamo detto che la posizione della memoria-schermo nella RAM può essere modificata per mezzo del registro di indirizzo 53272. Più precisamente i 4 bit più significativi (bit da 4 a 7) di questo registro sono utilizzati a questo scopo e il fatto di cambiare il loro valore (con una POKE) permette di cambiare la posizione della memoria-schermo.
La procedura da seguire è dunque la seguente:

```
POKE 53272, (PEEK(53272) AND 15) OR X
```

dove X può assumere i seguenti valori:

X	Bit 7654	Indirizzo di partenza della memoria-schermo
0	0 0 0 0	0 = $0000
16	0 0 0 1	1 024 = $0400
32	0 0 1 0	2 048 = $0800
48	0 0 1 1	3 072 = $0C00
64	0 1 0 0	4 096 = $1000
80	0 1 0 1	5 120 = $1400
96	0 1 1 0	6 144 = $1800
112	0 1 1 1	7 168 = $1C00
128	1 0 0 0	8 192 = $2000
144	1 0 0 1	9 216 = $2400
160	1 0 1 0	10 240 = $2800
176	1 0 1 1	11 264 = $2C00
192	1 1 0 0	12 288 = $3000
208	1 1 0 1	13 312 = $3400
224	1 1 1 0	14 336 = $3800
240	1 1 1 1	15 360 = $3C00

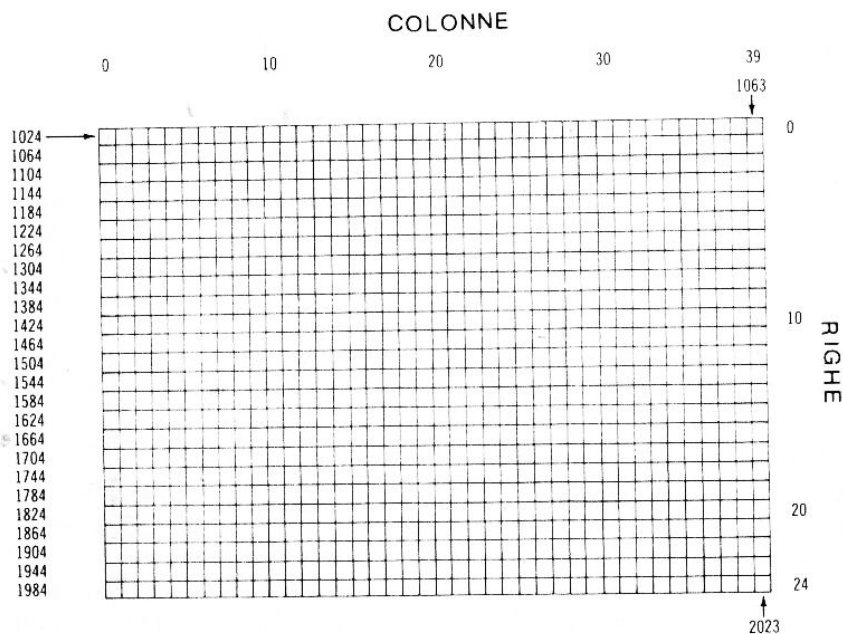
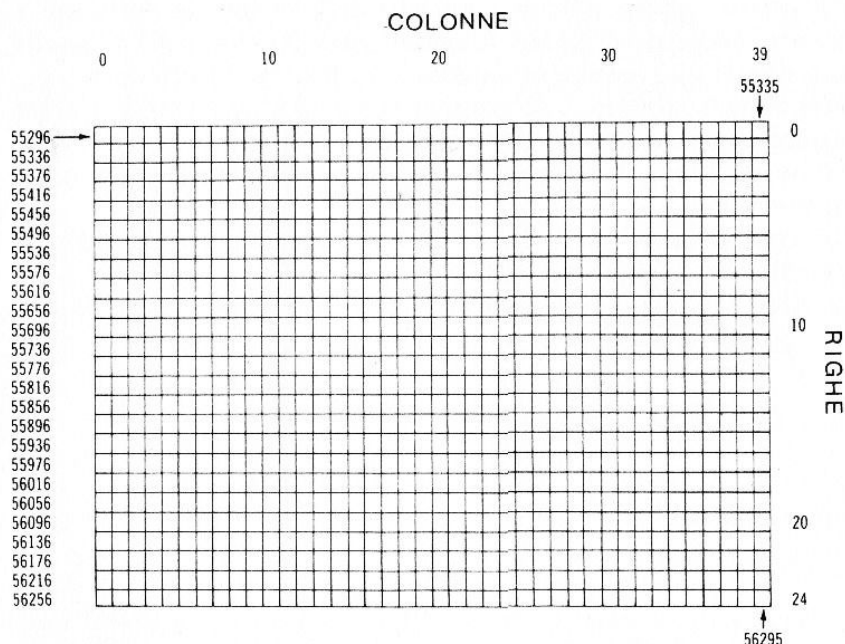
Questi indirizzi di partenza forniti si riferiscono al caso in cui viene selezionata la pagina n. 0. Quando ne viene selezionata un'altra, conviene aggiungere l'indirizzo di partenza di quest'ultima a quello della memoria-schermo.
Esempio:
Prendiamo X=144 e pagina n. 2.

L'indirizzo di partenza della memoria schermo sarà dunque: 32768+9216=41984=$A400.
Va notato che il valore di X che viene selezionato all'accensione della macchina è 16; questo comporta una memoria schermo posizionata tra gli indirizzi $0400 e $07E7, come abbiamo già visto.
Inoltre il sistema operativo del Commodore 64 e in particolare l'editore di testi deve conoscere la posizione della memoria-schermo. Questo può avvenire nel modo seguente.
Si determina subito: PAGE= indirizzo di partenza dalla memoria-schermo o diviso per 256 (dunque all'accensione sarà PAGE=4), poi si batte:

```
POKE 648, PAGE
```

2) La memoria-colore

Fino a questo punto abbiamo parlato della memoria destinata a conservare i codici dei caratteri da visualizzare. Abbiamo già detto che ognuno di essi può assumere un colore scelto fra 16 differenti.
Il colore di ogni carattere è determinato da un codice a 4 bit. Per poter immagazzinare il colore dei 1000 caratteri che costituiscono la visualizzazione standard, è necessaria una memoria speciale, chiamata memoria-colore.
Essa è composta di 1000 parole di 4 bit situate tra gli indirizzi 55296(=$D800) e 56295(=$DBE7).
Contrariamente alla memoria-schermo, la memoria-colore è fissa.
Diamo qui di seguito la mappa della memoria-schermo o della memoria-colore.

Memoria-schermo

Memoria-colore

Così se si vuole visualizzare in nero la lettera A alla riga 10, colonna 20, bisogna battere POKE 1444,01 e POKE 55716,0 (cfr. anche l'Appendice).

3) Il generatore di caratteri

La terza e ultima memoria necessaria per la rappresentazione in modo alfanumerico è il generatore di caratteri. In quest'ultimo sono immagazzinate le informazioni relative ad ogni carattere disponibile sul Commodore 64. Nella modalità standard, al momento dell'accensione, il controllore video va a cercare le informazioni che gli servono in un generatore di caratteri posto in una memoria ROM. Questa memoria a sola lettura contiene le informazioni relative ai caratteri maiusco-

li, minuscoli e semigrafici. Ogni carattere è definito da una matrice 8 x 8: per questo è necessario disporre per ognuno di essi di 8 byte di memoria. Esaminiamo la codifica della lettera A nel generatore di caratteri:

A è visualizzata sullo schermo da

```
· · · ● ● · · ·
· · ● ● ● ● · ·
· ● ● · · ● ● ·
· ● ● ● ● ● ● ·
· ● ● · · ● ● ·
· ● ● · · ● ● ·
· ● ● · · ● ● ·
· · · · · · · ·
```

a cui corrisponde nel generatore di caratteri

```
0 0 0 1 1 0 0 0 =  24
0 0 1 1 1 1 0 0 =  60
0 1 1 0 0 1 1 0 = 102
0 1 1 1 1 1 1 0 = 126
0 1 1 0 0 1 1 0 = 102
0 1 1 0 0 1 1 0 = 102
0 1 1 0 0 1 1 0 = 102
0 0 0 0 0 0 0 0 =   0
```

Il generatore di caratteri in ROM ha una capacità di 4 kbyte. Esistono dunque nel Commodore 64 due serie di 256 caratteri ciascuna, la prima comprende i caratteri maiuscoli e grafici, la seconda i caratteri maiuscoli, minuscoli e qualche carattere grafico.
Nell'appendice vengono forniti i codici dei differenti caratteri, così come sono visualizzati sullo schermo.
Diversamente dalla maggior parte dei microcomputer che esistono attualmente sul mercato, il generatore di caratteri del Commodore 64 (come d'altronde quello del VIC 20) è situato nello spazio indirizzabile del 6510 (il microprocessore che comanda la macchina): questo ci dà la possibilità di programmare noi stessi un nostro personale generatore di caratteri (a condizione però che sia realizzato nella memoria RAM di lettura/scrittura).
Il controllore video offre questa possibilità grazie ai bit 1, 2 e 3 del registro di indirizzo 53272(=$D018).
La tavola che segue fornisce la posizione del generatore di caratteri in funzione del valore assunto da questi tre bit.

X	Bit 3 2 1	Posizione del generatore di caratteri	Tipo di memoria
0	0 0 0	0 – 2 047 = $0000 – $07FF	RAM
2	0 0 1	2 048 – 4 095 = $0800 – $0FFF	RAM
4	0 1 0	4 096 – 6 143 = $1000 – $17FF	ROM (solo pagine 0 e 2)
6	0 1 1	6 144 – 8 191 = $1800 – $1FFF	ROM (solo pagine 0 e 2)
8	1 0 0	8 192 – 10 239 = $2000 – $27FF	RAM
10	1 0 1	10 240 – 12 287 = $2800 – $2FFF	RAM
12	1 1 0	12 288 – 14 335 = $3000 – $37FF	RAM
14	1 1 1	14 336 – 16 383 = $3800 – $3FFF	RAM

I valori riportati sopra sono dati per la pagina 0.
La ROM per la generazione dei caratteri risiede fisicamente tra gli indirizzi 53248E e 57343 ($D000 ÷ $DFFF). Ciò nonostante il Commodore 64 è progettato in maniera tale che il generatore sia "visto" ad altri indirizzi dello spazio indirizzabile del 6510. Così, quando viene selezionata la pagina n. 0, il generatore di caratteri in su ROM sarà visto come posizionato tra gli indirizzi $1000 e $1FFF. Ugualmente quando viene selezionata la pagina n. 2, esso sarà situato tra gli indirizzi 36864 e 40959 ($9000 ÷ $9FFF). Per selezionare dei caratteri maiuscoli e grafici, basterà battere sulla tastiera:

```
POKE 53272, (PEEK(53272) AND 240) OR 4
```

Analogamente per selezionare la serie dei caratteri maiuscoli e minuscoli, bisogna battere:

```
POKE 53272, (PEEK(53272) AND 240) OR 6
```

La tavola che segue fornisce in dettaglio il contenuto della ROM generatrice di caratteri.

Indirizzo fisico	Indirizzo di partenza pag. n°0	pag. n° 2	Caratteri selezionati
53 248 – 53 759	$1000	$9000	Maiuscoli
53 760 – 54 271	$1200	$9200	Grafici
54 272 – 54 783	$1400	$9400	Maiuscoli invertiti
54 784 – 55 295	$1600	$9600	Grafici invertiti
55 296 – 55 807	$1800	$9800	Minuscoli
55 808 – 56 319	$1A00	$9A00	Maiuscoli e grafici
56 320 – 56 831	$1C00	$9C00	Minuscoli invertiti
56 832 – 57 343	$1E00	$9E00	Maiuscoli e grafici invertiti

3.3 Generatore personalizzato di caratteri

Abbiamo visto in precedenza che il controllore video poteva accedere a un generatore di caratteri situato in RAM. E' dunque possibile a chiunque di programmare un proprio personale generatore di caratteri.
Prima di descrivere come ciò possa avvenire, dobbiamo fare alcune osservazioni importanti.
Quando si lavora in BASIC, il generatore di caratteri non può cominciare in una locazione qualsiasi della memoria. In particolare non può cominciare all'indirizzo 0 (cioè con i bit 3, 2 e 1 del registro di indirizzo 53272 uguali a "zero") poiché il Commodore 64 vi conserva certi dati e certi puntatori necessari per il suo funzionamento (la pagina 0 del microprocessore 6510 è posta tra gli indirizzi 0 e 255 e la catasta tra gli indirizzi 256 e 511). Inoltre non può cominciare all'indirizzo 2048 (bit 3, 2, 1 uguali a 0, 0, 1 rispettivamente) perché là cominciano i programmi BASIC.

Bisogna dunque definire il generatore di caratteri personale a partire dall'indirizzo 8192 (bit 3, 2, 1 uguali a 1, 0, 0 oppure 1, 0, 1 oppure 1, 1, 1).
Per fare iniziare il generatore di caratteri all'indirizzo 12288 ($3000), basterà scrivere:

```
POKE 53272, (PEEK(53272) AND 240) OR 12
```

Il generatore di caratteri occupa un certo spazio di memoria e questo può creare dei problemi nel caso di programmi piuttosto lunghi.
Come avete potuto notare, la posizione fisica del generatore di caratteri in ROM coincide parzialmente con l'indirizzo dei registri del controllore video (53248 ÷ 53294). Questo non è però un problema, poiché la ROM non è selezionata che quando il controllore video ha bisogno di accedere ai codici relativi a un carattere. Ciò nonostante, se, al momento della programmazione di un vostro generatore di caratteri, desiderate ricopiare alcuni (ad esempio le lettere), dovete fare una breve procedura preliminare.
Immettete da programma le due linee seguenti:

```
POKE 56334, PEEK(56334) AND 254
```

(che ha lo scopo di inibire le interruzioni, in particolare l'uso della tastiera) e

```
POKE 1, PEEK(1) AND 251
```

(che ha il ruolo di selezionare la ROM generatrice di caratteri e interdire l'accesso ai registri di ingresso/uscita).
Dopo di questo, copiate i caratteri che desiderate utilizzando il procedimento che descriveremo più avanti. Quando questo compito è esaurito, immettete da programma le due linee seguenti:

```
POKE 1, PEEK(1) OR 4
```

(che ha il compito di selezionare nuovamente i registri di controllo di ingresso/uscita — la ROM generatrice di caratteri non è più selezionata) e

```
POKE 56334, PEEK(56334) OR 1
```

(che ha il ruolo di autorizzare di nuovo le interruzioni, in particolare l'uso della tastiera).
Finora ci siamo volutamente limitati a descrivere l'aspetto generale del problema, tuttavia ampiamente sufficiente a chi voglia programmare in BASIC e forse in Assembler.
Quando si utilizza un generatore personalizzato di caratteri con l'interprete BASIC del Commodore 64, bisogna assicurarsi che esso non vada distrutto dal programma stesso oppure dalla memorizzazione di variabili e stringhe. Per fare questo può essere necessario proteggere una certa parte della RAM disponibile da operazioni di scrittura in BASIC.
Si può fare questo modificando i puntatori della fine della RAM e della zona di memorizzazione delle stringhe. Se si desidera memorizzare il generatore di caratteri a partire dall'indirizzo 12288 si potrà fare:

```
POKE 52,48 e  POKE 56,48
```

I due puntatori contengono ora l'indirizzo: 48 x 256 + 0 = 12288.
Finora abbiamo visto quanto è necessario fare per copiare in RAM certi caratteri della ROM. Illustreremo meglio l'argomento con un esempio di programma. Lo scopo è di conservare le lettere maiuscole e di programmare come vogliamo i caratteri grafici. Faremo riferimento alla tavola dei codici-schermo riportata in Appendice.
La serie di caratteri n. 1 inizia con il segno@, poi fornisce i codici delle lettere maiuscole. Inoltre sappiamo che questa serie di caratteri inizia all'indirizzo 53248 e che ciascun carattere occupa 8 byte.
Dunque le lettere dell'alfabeto saranno situate tra gli indirizzi 53256 e 53256 + 26 x 8 = 53464 e saranno conservate tra gli indirizzi 12296 e 12504.
Ricapitolando tutto quanto abbiamo detto, possiamo scrivere il programma che segue:

```
 10 POKE 52,48: POKE 56,48
 20 POKE 56334, PEEK(56334) AND 254
 30 POKE 53272, (PEEK(53272) AND 240) OR 12
 40 POKE 1, PEEK(1) AND 251
 50 FOR I = 0 TO 207
 60 POKE I + 12296, PEEK(I + 53256)
 70 NEXT I
 80 POKE 1, PEEK(1) OR 4
 90 POKE 56334, PEEK(56334) OR 1
100 END
```

Dopo aver immesso da tastiera questo programma, fate RUN e poi provate subito a battere sui tasti che rappresentano le lettere: non deve essere cambiato nulla.
Se invece battete un qualsiasi altro tasto dovete veder apparire casualmente i caratteri sullo schermo poiché il contenuto del generatore di caratteri su RAM è aleatorio.
Ora tentiamo di sostituire il carattere @ con il carattere © che indica il Copyright.
Questo carattere è rappresentato sullo schermo nella maniera seguente:

	che, in binario, nel generatore diventa	
· · ● ● ● ● · ·		0 0 1 1 1 1 0 0 = 60
· ● · · · · ● ·		0 1 0 0 0 0 1 0 = 66
● · · ● ● · · ●		1 0 0 1 1 0 0 1 = 153
● · ● · · · · ●		1 0 1 0 0 0 0 1 = 161
● · ● · · · · ●		1 0 1 0 0 0 0 1 = 161
● · · ● ● · · ●		1 0 0 1 1 0 0 1 = 153
· ● · · · · ● ·		0 1 0 0 0 0 1 0 = 66
· · ● ● ● ● · ·		0 0 1 1 1 1 0 0 = 60

Per creare il carattere e inserirlo nella RAM basta battere il programmino seguente:

```
10 POKE 53272, (PEEK(53272) AND 240) OR 12
20 DATA 60, 66, 153, 161, 161, 153, 66, 60
30 FOR I = 0 TO 7: READ A
40 POKE 12288 + I, A: NEXT I
```

3.4 Modo di visualizzazione alfanumerico a più colori

1) Caratteri a più colori

Abbiamo visto in precedenza che nella modalità alfanumerica standard ogni carattere (composto da 8 x 8 = 64 punti distinti) può assumere un colore scelto tra 16 possibili. Ogni carattere è rappresentato da un codice-schermo che corrisponde a 8 byte immagazzinati in un generatore di caratteri su ROM o su RAM. Nel modo alfanumerico standard ogni carattere è o acceso o spento. Se è acceso, il suo colore corrisponde al codice-colore conservato nella locazione della memoria del colore relativo al carattere considerato. Si è spento, il suo colore corrisponde a quello dello sfondo dello schermo.
Nel modo a più colori ogni punto di ogni matrice 8 x 8 può prendere quattro colori diversi invece di due, come nel caso precedente. Questo si traduce in una perdita di risoluzione orizzontale della rappresentazione (la risoluzione è dimezzata).
I quattro colori possibili sono allora i seguenti:

— colore dello schermo (colore immagazzinato nel registro del colore di sfondo n. 0)
— colore immagazzinato nel registro del colore di sfondo n. 1
— colore immagazzinato nel registro del colore di sfondo n. 2
— colore del carattere

Selezionare questa modalità di visualizzazione è estremamente semplice: a questo scopo basta dare il valore 1 al bit 4 del registro di indirizzo 53270 (=$D016) presente sul controllore video.

Si batte poi sulla tastiera:

```
POKE 53270, PEEK(53270) OR 16
```

Per tornare alla modalità di rappresentazione standard basta battere:

```
POKE 53270, PEEK(53270) AND 239
```

Questo modo di visualizzazione dà accesso a una rappresentazione di tipo semigrafico a più colori (e quindi anche alfanumerico). Esso può venire facilmente mescolato con una visualizzazione di tipo grafico ad alta risoluzione di cui parleremo nel prossimo paragrafo.

Ogni posizione dello schermo corrispondente a un carattere può essere programmata sia in modalità semigrafica a più colori sia in modalità grafica ad alta risoluzione. Questo avviene nella maniera seguente. Abbiamo visto che la memoria-colore è composta di 1000 parole di 4 bit posta tra gli indirizzi 55296 e 56295. Se il codice-colore relativo a un carattere è superiore a 8 (bit 3 = 1), allora questo carattere sarà rappresentato in modalità a più colori. Se invece il codice colore è inferiore a 7, lo spazio corrispondente sarà rappresentato in modalità grafica ad alta risoluzione con il colore scelto (colore da 0 a 7).
In altre parole, se il bit 3 del codice-colore di un carattere vale "zero", quest'ultimo sarà visualizzato "normalmente" sullo schermo. Invece se il bit 3 vale "uno", il carattere sarà visualizzato in modalità a più colori.
Consideriamo ad esempio la lettura A la cui rappresentazione nel generatore di caratteri è la seguente:

```
· · · ● ● · · ·                           0 0 0 1 1 0 0 0
· · ● ● ● ● · ·                           0 0 1 1 1 1 0 0
· ● ● · · ● ● ·                           0 1 1 0 0 1 1 0
· ● ● ● ● ● ● ·    cui corrisponde        0 1 1 1 1 1 1 0
· ● ● · · ● ● ·       in binario          0 1 1 0 0 1 1 0
· ● ● · · ● ● ·                           0 1 1 0 0 1 1 0
· ● ● · · ● ● ·                           0 1 1 0 0 1 1 0
· · · · · · · ·                           0 0 0 0 0 0 0 0
```

Per la determinazione del colore di ciascun punto è necessario raggruppare a coppie i bit del generatore di caratteri. Avremo:

```
0 0   0 1   1 0   0 0
0 0   1 1   1 1   0 0
0 1   1 0   0 1   1 0
0 1   1 1   1 1   1 0
0 1   1 0   0 1   1 0
0 1   1 0   0 1   1 0
0 1   1 0   0 1   1 0
0 0   0 0   0 0   0 0
```

Ciascuna coppia di bit assume uno e un solo colore a seconda del suo valore. La tavola che segue permette di determinare questo colore.

Coppia di bit	Colore determinato da	Indirizzo
0 0	Registro del colore di sfondo n. 0 (colore schermo)	53 281(= $D021)
0 1	Registro del colore di sfondo n. 1	53 282(= $D022)
1 0	Registro del colore di sfondo n. 2	53 283(= $D023)
1 1	Colore del carattere (3 bit meno significativi del codice-colore contenuto nella memoria-colore)	RAM colore

La tavola dei codici-colore è contenuta in Appendice.
Per ottenere la lettera A composta con i colori seguenti al centro dello schermo (per esempio, indirizzo 1524), cioè:

```
B B V V R R B B
B B N N N N B B          B = Bianco
V V R R V V R R    con   R = Rosso
V V N N N N R R          V = Verde
V V R R V V R R          N = Nero
V V R R V V R R
V V R R V V R R
B B B B B B B B
```

basterà comporre il piccolo programma che segue.

```
10 POKE 53270, PEEK(53270) OR 16
20 POKE 53281, 1
30 POKE 53282, 5
40 POKE 53283, 2
50 POKE  1524, 1
60 POKE 55796, 8
```

2) Sfondo a più colori

Ritorniamo ora alla modalità alfanumerica standard e consideriamone una variante.
Abbiamo visto che ogni carattere può essere rappresentato con un colore il cui codice si trova nella memoria-colore, mentre lo schermo ha un colore di sfondo programmabile grazie al registro di indirizzo 53281 (=$D021).
In questo caso, il colore dello sfondo è il medesimo per ogni carattere. Tuttavia è anche possibile, grazie a una nuova modalità di visualizzazione ottenibile con il controllore video, ottenere un colore di sfondo diverso per ogni carattere rappresentato. Ben inteso, esistono delle limitazioni per questa modalità di rappresentazione: sono utilizzabili solo i primi 64 caratteri del generatore (ROM serie n. 1, n. 2 o RAM). Questo avviene perché due dei bit che compongono il codice-schermo del carattere visualizzato sono utilizzati per selezionare il colore dello sfondo. Il colore di ogni carattere è, al solito, immagazzinato nella memoria-colore.
Questa modalità di visualizzazione è selezionata attribuendo il valore 1 al bit 6 del registro di indirizzo 53265(=$D011). Si deve battere sulla tastiera:

```
POKE 53265, PEEK(53265) OR 64
```

Per tornare alla rappresentazione standard, basta fare:

```
POKE 53265, PEEK(53265) AND 191
```

Inoltre quattro registri permettono di selezionare il colore dello sfondo di ogni carattere.

Numero del carattere	Registro di selezione
0-63	53 281 (=$D021)
64-127	53 282 (=$D022)
128-191	53 283 (=$D023)
192-255	53 284 (=$D024)

Abbiamo detto che unicamente i primi 64 caratteri di ogni serie possono essere visualizzati. Perciò i quattro sottoinsiemi dei caratteri mostrati nella tavola precedente (0—63, 64—127, 128—191, 192—255) rappresenteranno simboli sullo schermo ma con colori di sfondo diversi, selezionabili da uno dei registri con indirizzi da 53281 e 53284. Ad esempio, per ottenere un carattere A bianco su fondo nero si più scrivere:

```
10 POKE 1564, 1
20 POKE 53281, 1
30 POKE 55836, 0
```

Abbiamo dunque terminato con i diversi modi di visualizzazione alfanumerica. Possiamo ora affrontare la grafica ad alta risoluzione.

3.5 Grafica ad alta risoluzione

La grafica ad alta risoluzione è forse una delle caratteristiche più interessanti del Commodore 64. Con questa modalità di rappresentazione sullo schermo è possibile ottenere una risoluzione di 320 per 200 punti aventi due colori diversi oppure una risoluzione di 160 per 200 punti con quattro colori. Se avete ben capito come utilizzare il generatore di caratteri programmabile, non dovreste aver problemi ad ottenere dei magnifici grafici ad alta risoluzione sul video del vostro Commodore.

3.5.1 Rappresentazione a due colori

Abbiamo visto in precedenza che nella modalità standard di rappresentazione, il Commodore 64 permette di visualizzare sullo schermo 25 righe di 40 caratteri ognuna.
Ognuno di questi è formato per mezzo di una matrice di punti formato 8 x 8. Questo significa che la risoluzione orizzontale di visualizzazione è uguale a 40 x 8 = 320 punti. Analogamente la risoluzione verticale di visualizzazione vale 25 x 8 = 200 punti. La rappresentazione ad alta risoluzione consiste semplicemente nell'accendere o nello spegnere ognuno di questi punti. Sono dunque disponibili 64000 punti diversi. Se si considera che c'è bisogno di un bit (con valore 1 o 0) per

visualizzare un punto, la memoria necessaria per coprire tutto lo schermo sarà di 64000/8 = 8000 byte.
Per ottenere una rappresentazione ad alta risoluzione sul Commodore 64 è necessario:

— riservare una memoria-schermo da 8 kbyte
— programmare interamente il generatore di caratteri.

1) Linguaggio di programmazione

Prima di andare più oltre nella descrizione di questa procedura, va precisato un punto. Tutti i capitoli che compongono questo libro utilizzano esempi scritti in BASIC.
Questo è sufficiente nella stragrande maggioranza dei casi, ma non per la rappresentazione ad alta risoluzione. Come comprenderete, le procedure di inizializzazione della memoria-schermo e di programmazione del generatore di caratteri sono di una lentezza esasperante quando sono realizzate in BASIC.
La soluzione consiste dunque nel passare all'Assembler, soprattutto se si desidera produrre delle animazioni analoghe a quelle utilizzate nei videogame.
Per approfondire questi argomenti ci ripromettiamo di scrivere ancora nella seconda parte di quest'opera. Ritorniamo dunque al nostro argomento.

2) Selezione della grafica ad alta risoluzione

Innanzitutto è necessario selezionare questa modalità di rappresentazione. Si può fare questo in una maniera analoga a quella che abbiamo già visto prima: basta posizionare a 1 il bit 5 del registro di indirizzo 53265 (=$D011) presente sul controllore video.
Si batte dunque:

```
POKE 53265, PEEK(53265) OR 32
```

Per ritornare nella modalità standard di rappresentazione basta comporre:

```
POKE 53265, PEEK(53265) AND 223
```

Al paragrafo 3.3 di questo stesso capitolo abbiamo visto che è possibile programmarci da soli il nostro generatore di caratteri. Nel caso di una rappresentazione standard, la memoria-schermo contiene dei codici corrispondenti a diverse posizioni nel generatore di caratteri. In questo caso è il contenuto stesso di quest'ultimo che sarà mostrato sullo schermo: la memoria-schermo sarà allora utilizzata per immagazzinare il colore della rappresentazione (le informazioni che riguardano il colore della visualizzazione in questo caso non provengono più dalla memoria-colore). Nella modalità standard di rappresentazione, quando si fa POKE 1024,1 si vede apparire la lettera A maiuscola in alto a destra sullo schermo. In questo caso immette un valore all'indirizzo 1024 permette di determinare il colore degli 8 x 8 = 64 bit posti in alto a destra sullo schermo.
Utilizzando le locazioni di memoria con indirizzi 1024 ÷ 2023 è dunque possibile programmare il colore di ogni punto della visualizzazione ad alta risoluzione con il metodo seguente:

— i 4 bit meno significativi di ogni locazione di memoria permettono di programmare il colore di tutti i bit della matrice 8 x 8 corrispondente, che sono messi a 0 (spenti)
— i 4 bit più significativi di ogni locazione di memoria permettono di programmare il colore di tutti i bit della matrice 8 x 8 corrispondente, che assumono il valore 1 (accesi).

Ogni matrice di punti 8 x 8 può dunque essere composta di punti di due colori differenti, scelti fra 16 possibili.
Abbiamo già visto come determinare la modalità di rappresentazione ad alta risoluzione. E' necessario però determinare anche la posizione del generatore di caratteri. Questo deve essere residente in RAM e deve contenere 8 kbyte. Basta dunque scrivere:

```
POKE 53272, PEEK(53272) OR 8
```

per ottenere un generatore di caratteri che inizia all'indirizzo 8192 e finisce all'indirizzo 16383.
Ricapitolando quanto abbiamo visto, il programmino che segue permette di selezionare la rappresentazione ad alta risoluzione:

```
10 POKE 53265, PEEK(53265) OR 32
20 POKE 53272, PEEK(53272) OR 8
```

Dopo averlo introdotto, battete RUN: vedrete apparire una cosa qualsiasi sullo schermo, ma ciò è normale poiché il contenuto della RAM del Commodore 64 è casuale al momento dell'accensione.

3) Inizializzazione

Per rimediare a questo inconveniente e ottenere una rappresentazione totalmente inizializzata (ad esempio completamente bianca) basta riempire subito tutte le locazioni di memoria che compongono il generatore di caratteri con il valore 0 e poi riempire la memoria-colore (ex memoria-schermo) con il valore 1 (per avere il bianco). Basta dunque aggiungere al programma precedente le seguenti istruzioni:

```
30 FOR I = 8192 TO 16383
40 POKE I, 0: NEXT
50 FOR I = 1024 TO 2023
60 POKE I, 1: NEXT
```

Vediamo ora in quale maniera è possibile accendere o spegnere un punto sullo schermo.

4) Visualizzazione

E' prima di tutto necessario localizzare il punto considerato, cioè:

— determinare a quale matrice 8 x 8 appartiene
— individuare poi quel punto nella matrice.

Per fare ciò utilizzeremo le coordinate X e Y del punto considerato. E' chiaro che la sua ascissa X dovrà essere compresa fra 0 e 319, mentre l'ordinata Y sarà compresa fra 0 e 199. Queste coordinate corrispondono a un "carattere" (matrice 8 x 8) la cui posizione orizzontale (colonna) sarà compresa fra 0 e 39 e la cui posizione verticale (riga) sarà compresa fra 0 e 24, poiché la rappresentazione standard possiede 25 righe di 40 colonne.

Lo schema che segue fornisce l'organizzazione della memoria generatrice di caratteri.

	COLONNA 0	COLONNA 1		COLONNA 39
	BYTE 0	BYTE 8	...	BYTE 312
	BYTE 1	BYTE 9	...	BYTE 313
	BYTE 2	BYTE 10	...	BYTE 314
RIGA 0	BYTE 3	BYTE 11	...	BYTE 315
	BYTE 4	BYTE 12	...	BYTE 316
	BYTE 5	BYTE 13	...	BYTE 317
	BYTE 6	BYTE 14	...	BYTE 318
	BYTE 7	BYTE 15	...	BYTE 319
	BYTE 320	BYTE 328	...	BYTE 632
	BYTE 321	BYTE 329	...	BYTE 633
	BYTE 322	BYTE 330	...	BYTE 634
RIGA 1	BYTE 323	BYTE 331	...	BYTE 635
	BYTE 324	BYTE 332	...	BYTE 636
	BYTE 325	BYTE 333	...	BYTE 637
	BYTE 326	BYTE 334	...	BYTE 638
	BYTE 327	BYTE 335	...	BYTE 639
	⋮			
	BYTE 7680	BYTE 7688	...	BYTE 7992
	BYTE 7681	BYTE 7689	...	BYTE 7994
	BYTE 7682	BYTE 7690	...	BYTE 7995
RIGA 24	BYTE 7683	BYTE 7691	...	BYTE 7996
	BYTE 7684	BYTE 7692	...	BYTE 7997
	BYTE 7685	BYTE 7693	...	BYTE 7998
	BYTE 7686	BYTE 7694	...	BYTE 7998
	BYTE 7687	BYTE 7695	...	BYTE 7999

Poiché X e Y sono le coordinate del punto considerato, si potrà determinare:

RIGA = INT (Y/8)

(il numero della riga in cui si trova il punto, compreso fra 0 e 24), nonché:

COLONNA = INT(X/8)

(il numero della colonna in cui si trova il punto, compreso fra 0 e 39). Questo ci fornisce la posizione del "carattere" (matrice 8 x 8) sul video. Ora per conoscere la posizione esatta del bit è necessario conoscere la riga in cui si trova (una riga in rappresentazione standard vale 8 righe ad alta risoluzione).
Si avrà allora:

LC = Y AND 7

Il numero di byte-memoria interessato sarà dunque (se ci si riferisce allo schema precedente):

BYTE = 8192 +320*RIGA +8*COLONNA + LC

Per poter visualizzare un solo punto è necessario conoscere la posizione esatta del bit interessato nel byte. Esso sarà determinato da:

BIT = 7 – (X AND 7)

Per visualizzare il punto, basta battere sulla tastiera:

POKE BYTE, PEEK(BYTE) OR 2 ↑ BIT

Per spegnere il punto, si può battere:

POKE BYTE, PEEK(BYTE) AND (255 — 2 ↑ BIT)

Ad esempio, volendo visualizzare il punto di coordinate X = 120 e Y = 98 si ha:

RIGA = INT(98/8) = 12
COLONNA = INT(120/8) = 15
LC = 98 AND 7 = 2
BYTE = 8192 + 320*12 + 8*15 + 2 = 12154
BIT = 7 – (120 AND 7) = 7

e si batte

```
POKE 12154, PEEK(12154) OR 128
```

per accenderlo.

5) Esempio applicativo

Per illustrare quanto abbiamo detto in questo paragrafo, diamo un esempio di programma per il tracciamento di una curva ad alta risoluzione.
Il listing del programma è il seguente:

```
 10 REM PROGRAMMA TRACCIAMENTO
 20 DEF FNA(X) = SIN(X)/X
 30 INPUT "INTERVALLO?"; X1, X2
 40 PA = (X2 - X1)/320
 50 MI = 1.7 E 38: MA = -1.7 E 38
 60 FOR X = 0 TO 319
 70 X3 = X1 + X*PA
 80 IF X3 = 0 THEN Y = 1: GO TO 100
 90 Y = FNA(X3)
100 IF Y < MI THEN MI = Y
110 IF Y > MA THEN MA = Y
120 NEXT
130 B = MA*199/(MA - MI): A = 199/(MI - MA)
140 POKE 53265, PEEK(53265) OR 32
150 POKE 53272, PEEK(53272) OR 8
160 FOR I = 8192 TO 16383
170 POKE I, 0: NEXT
180 FOR I = 1024 TO 2023
190 POKE I, 1: NEXT
200 FOR X = 0 TO 319
210 X3 = X1 + X*PA
220 IF X3 = 0 THEN Y = INT(A + B): GO TO 240
230 Y = INT (B + A*FNA(X3))
240 LI = INT(Y/8)
250 CO = INT(X/8)
260 LC = Y AND 7
270 OC = 8192 + 320*LI + 8*CO + LC
280 BI = 7 - (7 AND X)
290 POKE OC, PEEK(OC) OR (2↑BI)
300 NEXT X
310 GO TO 310
```

La funzione da tracciare deve essere introdotta alla linea 20 sotto forma di DEF FN.
Dopo aver battuto RUN, immettete nel programma i limiti inferiore e superiore dell'intervallo di studio della funzione.
Poiché è necessario che la funzione sia definita nell'intervallo scelto sono state introdotte nel programma le linee 80 e 220 che permettono di tener conto che SIN(X)/X = 1 per X = 0.
Per tracciare curve di altro tipo, conviene eventualmente modificare il programma perché la funzione corrispondente sia sempre definita.
Per tornare al nostro programma, la curva viene tracciata punto a punto con un passo PA uguale a X2—X1/320, in cui il numero 320 corrisponde naturalmente alla risoluzione della rappresentazione secondo l'asse X.
La curva viene raffigurata automaticamente nella giusta scala poiché il programma determina il minimo (MI) e il massimo (MA) della funzione nell'intervallo stabilito.
Quando farete girare il programma, vi renderete conto della sua lentezza esasperante. La soluzione a questo tipo di problema è, come abbiamo già detto, l'Assembler.
A questo punto abbiamo finito con la rappresentazione ad alta risoluzione a due colori. Siamo però pronti ad affrontare la variante costituita dalla rappresentazione ad alta risoluzione a più colori.

Curva ad alta risoluzione.

3.5.2 Rappresentazione a più colori

Questa modalità di rappresentazione grafica su video deriva dalla rappresentazione alfanumerica a più colori che abbiamo già illustrato. Ogni punto dello schermo contenuto in una matrice 8 x 8 può assumere quattro colori differenti selezionati da:

— registro del colore di sfondo n. 0
— memoria-schermo (indirizzi 1024÷2023)
— memoria-colore (indirizzi 55296÷56295).

La memoria-schermo permette di selezionare due colori diversi esattamente come nel caso della rappresentazione ad alta risoluzione a due colori.
La memoria-colore permette di selezionare un quarto colore grazie alle sue 1000 parole di 4 bit.
Questa modalità di visualizzazione viene selezionata attribuendo il valore 1 al bit 5 del registro di indirizzo 53265 e al bit 4 del registro di indirizzo 53270.
Per selezionare questa modalità bisogna dunque battere sulla tastiera:

```
POKE 53265, PEEK(53265) OR 32
POKE 53270, PEEK(53270) OR 16
```

Per ritornare alla rappresentazione standard, bisogna battere:

```
POKE 53265, PEEK(53265) AND 223
POKE 53270, PEEK(53270) AND 239
```

Come nel caso della rappresentazione alfanumerica a più colori, la risoluzione orizzontale è dimezzata (160 x 200 punti) poiché un punto sullo schermo corrisponde a un bit di memoria.
La determinazione del colore dei punti all'interno della matrice 8 x 8 avviene associandoli per coppie, come abbiamo già visto al cap. 2.

Coppia di bit	Colore determinato da	Indirizzo
0 0	Colore di sfondo	53 281(= $D021)
0 1	4 bit più significativi in memoria-schermo	1 024 ÷ 2 023($0400÷$07E7)
1 0	4 bit meno significativi in memoria-schermo	1 024 ÷ 2023($0400÷$07E7)
1 1	Memoria-colore	55 296÷56 295($D800÷$DBE7)

A questo punto abbiamo terminato con le modalità di rappresentazione alfanumerica e grafica.
Possiamo ora affrontare una delle caratteristiche più originali del Commodore 64, cioè gli SPRITE.

3.6 Gli SPRITE

Gli SPRITE non sono altro che caratteri speciali di dimensioni più grandi, definibili dall'utilizzatore, che possono essere visualizzati in qualsiasi posizione dello schermo. Conoscete tutti i piccoli mostri (tipo PAC-MAN) che popolano i videogame. Il Commodore 64 permette a tutti, anche a voi, di programmarsi questi giochi e di utilizzare per questo anche solo il BASIC, poiché gli SPRITE sono gestiti automaticamente dal circuito di controllo video.
Per definire un vostro "personaggio" basterà, ad esempio, indicare al microcomputer la sua forma, il suo colore, la sua posizione sul video ...ed è tutto, o quasi.
Il Commodore 64 permette normalmente di visualizzare 8 SPRITE (numerati da 0 a 7) simultaneamente; questo avviene con qualsiasi modalità di rappresentazione (alfanumerica, grafica ad alta risoluzione, a più colori, ecc.).
Ognuno degli SPRITE gestiti dal controllore video possiede le seguenti caratteristiche:

— è definito da una matrice di punti formato 24 (orizzontali) x 21 (verticali)
— ha un suo proprio colore
— può essere rappresentato a più colori (con risoluzione orizzontale dimezzata)
— le sue dimensioni possono essere raddoppiate (in altezza, in larghezza o in entrambe le direzioni)
— quando viene visualizzato può sia nascondere quanto si trova sullo schermo dietro di lui sia apparire in sovraimpressione (priorità SPRITE-SFONDO)
— può nascondere un altro SPRITE o sparire dietro di esso (priorità SPRITE-SPRITE)
— può rivelare una collisione con lo sfondo dello schermo.

Esamineremo in dettaglio queste diverse caratteristiche. La programmazione di queste funzioni fa appello ai numerosi registri interni che sono presenti sul controllore video.

1) Definizione di uno SPRITE

Definire uno SPRITE è altrettanto semplice che definisce un carattere programmabile (come abbiamo già visto). L'unica differenza è che uno SPRITE è definito da una matrice di punti di formato 24 x 21 (invece che 8 x 8). Ognuno di essi necessita dunque di 504 bit di memoria, che corrispondono a 63 byte. Questi sono organizzati secondo 21 righe di 3 colonne, come vediamo qui sotto.

```
BYTE 0      BYTE 1      BYTE 2
BYTE 3      BYTE 4      BYTE 5
BYTE 6      BYTE 7      BYTE 8
  .           .           .
  .           .           .
  .           .           .
  .           .           .
  .           .           .
BYTE 60     BYTE 61     BYTE 62
```

Inoltre è necessario un 64.mo byte per conoscere l'indirizzo in memoria in cui si trova la definizione dello SPRITE (puntatore di SPRITE).

In uno SPRITE ogni bit di valore 1 sarà rappresentato sul video nel colore desiderato (colore dello SPRITE), mentre ogni bit di valore 0 sarà trasparente e lascerà vedere quanto era già visualizzato in precedenza.
Poiché ad ogni SPRITE è associato un puntatore (come visto prima) ne esistono dunque 8 che sono posti negli ultimi 8 byte dello spazio di 1 kbyte riservato alla memoria-schermo (in realtà 1 k equivale e 1024 byte). Normalmente (memoria-schermo situata tra gli indirizzi 1024 e 2023), questi 8 puntatori sono situati agli indirizzi 2040 — 2047 ($07F8 ÷ $07FF).
Ben inteso, se la posizione della memoria schermo cambia, cambierà anche l'indirizzo dei puntatori.
Il loro contenuto è un numero compreso tra 0 e 255; questo permette di posizionare la definizione di ogni SPRITE a un indirizzo qualsiasi del blocco di 16 kbyte indirizzabile da parte del controllore video (256 x 64 byte = kbyte).
Per esempio, se il puntatore dello SPRITE n. 3, posto all'indirizzo 2043, contiene il valore 56, questo significa che la sua definizione è posizionata fra gli indirizzi 56 x 64 = 3584 e 3646. Oltre a questo, bisogna naturalmente tener conto dello spostamento corrispondente alla pagina di 16 kbyte considerata (pagina da 0 a 3).
Insomma uno SPRITE viene definito a partire dall'indirizzo seguente:

```
INDIRIZZO = (NUMERO DI PAGINA*16384)+
+(INDIRIZZO FORNITO DAL PUNTATORE)*64
```

Bisogna fare attenzione, quando si utilizzano le pagine da 0 a 2 di non far coincidere gli indirizzi di definizione degli SPRITE con la posizione del generatore di caratteri in ROM (indirizzi 4096 ÷ 8191 per la pagina 0 e 36864 ÷ 40959 per la pagina 2).

2) Visualizzazione di uno SPRITE

Ora che sappiamo come definire gli SPRITE, vediamo come ottenere la loro visualizzazione. Un registro speciale del controllore video permette di selezionare quale (o quali) SPRITE saranno visualizzati. Esso è posizionato all'indirizzo 53269 (=$D015) e ogni bit da 1 a 7 permette di "accendere" o di "spegnere" lo SPRITE corrispondente.

Ad esempio il bit 1 permette di comandare lo SPRITE n. 1. Quando tale bit assume il valore 1, lo SPRITE verrà acceso. Quando invece assume il valore 0, lo SPRITE verrà spento.
Più in generale, per accendere lo SPRITE numero N, basta battere:

```
POKE 53269, PEEK(53269) OR (2 ↑ N) con 0 ≤ N ≤ 7
```

Per spegnerlo si batte:

```
POKE 53269, PEEK(53269) AND (255 – 2 ↑ N)
```

3) Colore di uno SPRITE

Esistono anche 8 registri che permettono di definire il colore degli SPRITE. Essi sono posizionati tra gli indirizzi 53287(=$D027) e 53294(=$D02E) e permettono rispettivamente di selezionare il colore degli SPRITE da 0 a 7. Ciascuno di essi può assumere uno dei 16 colori possibili.
Per esempio, per avere lo SPRITE n. 2 di colore rosso basta battere:

```
POKE 53289, 2
```

4) SPRITE a più colori

I registri prima descritti permettono di selezionare il colore degli SPRITE standard che possiedono una risoluzione di 24 x 21 punti. Così come esiste una modalità di rappresentazione alfanumerica a più colori, è possibile anche ottenere degli SPRITE a più colori in cambio di un dimezzamento della risoluzione orizzontale.
Questo può avvenire per mezzo del registro 53276 (=$D01C) i cui bit da 0 a 7 permettono rispettivamente di selezionare la modalità a più colori per gli SPRITE da 0 a 7.
Attribuendo il valore 1 a un bit di questo registro si può selezionare la modalità a più colori per lo SPRITE corrispondente. Inizializzandolo

nuovamente col valore 0 si può ritornare al funzionamento normale.
L'istruzione seguente

```
POKE 53276, PEEK(53276) OR (2 ↑ N)
```

mette lo SPRITE numero N in modalità a più colori.
Analogamente:

```
POKE 53276, PEEK(53276) AND (255 – 2 ↑ N)
```

permette di ritornare al funzionamento normale per lo SPRITE numero N.
Questa modalità a più colori consente 4 colori diversi per ogni SPRITE. Poiché la loro risoluzione orizzontale è dimezzata, i bit devono essere associati per coppie, proprio come per le rappresentazioni alfanumeriche a più colori e per le rappresentazioni ad alta risoluzione a più colori.
La tavola che segue ricapitola i colori assunti da un punto di uno SPRITE in funzione del valore dei 2 bit corrispondenti.

Coppia di bit	Colore determinato da	Indirizzo
0 0	Colore schermo (SPRITE trasparente)	
0 1	Registro a più colori n. 0	53 285 (= $D025)
1 0	Registro del colore dello SPRITE	53 287÷53 294 ($D027÷$D02E)
1 1	Registro a più colori n. 1	53 286 (= $D026)

5) Dimensione dello SPRITE

L'ultima funzione concernente l'inizializzazione di uno SPRITE che ci rimane da illustrare è la dimensione. Abbiamo già visto che è possibile moltiplicare per due le dimensioni orizzontali o verticali (o entrambe contemporaneamente).

Questo avviene per mezzo dei due registri di indirizzo 53277(=$D01D) e 53271(=$D017).

Attribuire il valore 1 a un bit del registro di indirizzo 53277 permette di moltiplicare per due la dimensione dello SPRITE corrispondente lungo l'asse X (direzione orizzontale).
Posizionare a 0 un bit del registro di indirizzo 53277 permette di tornare alla dimensione normale per lo SPRITE corrispondente.
Il registro di indirizzo 53271 funziona in modo analogo, ma lungo l'asse Y (direzione verticale).
Di conseguenza, si scrive:

```
POKE 53277, PEEK(53277) OR (2 ↑ N)
```

per raddoppiare la dimensione dello SPRITE numero N lungo l'asse X;

```
POKE 53271, PEEK(53271) OR (2 ↑ N)
```

per raddoppiare la dimensione dello SPRITE numero N lungo l'asse Y;

```
POKE 53277, PEEK(53277) AND (255 – 2 ↑ N)
```

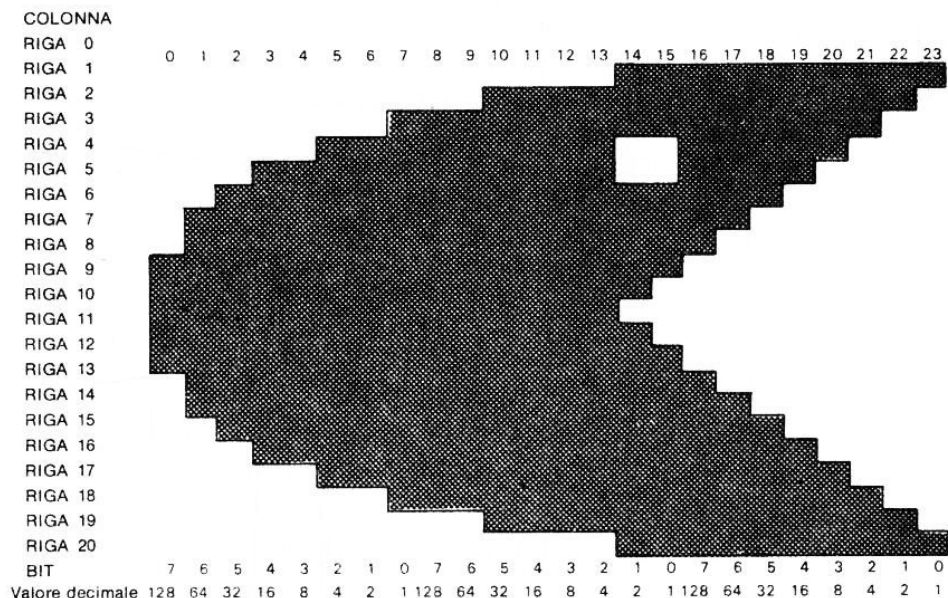
perché lo SPRITE numero N torni alla dimensione standard lungo l'asse X;

```
POKE 53271, PEEK(53271) AND (255 – 2 ↑ N)
```

perché lo SPRITE numero N torni alla dimensione standard lungo l'asse Y.
Va notato che l'aumento della dimensione di uno SPRITE non ne modifica la risoluzione di visualizzazione.

6) Un esempio

Prima di passare al prossimo paragrafo che tratta del posizionamento degli SPRITE sullo schermo, illustreremo un esempio in tutti i dettagli. Vogliamo realizzare quella figura ben nota come PAC-MAN.
Per fare questo utilizziamo una tabella che ci permette di conoscere

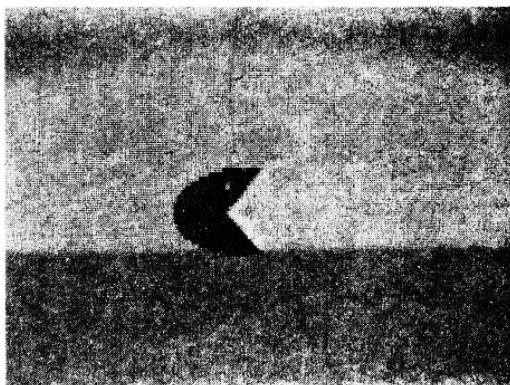
direttamente i valori da caricare nei 63 byte necessari alla definizione di uno SPRITE.

A partire da questo disegno deduciamo i valori da caricare nei 63 byte.

BYTE 0 = 0
BYTE 1 = 3
BYTE 2 = 255
BYTE 3 = 0
BYTE 4 = 63
BYTE 5 = 254
BYTE 6 = 1
BYTE 7 = 255
BYTE 8 = 252
BYTE 9 = 7
BYTE 10 = 252
BYTE 11 = 248
BYTE 12 = 31
BYTE 13 = 252
BYTE 14 = 240
BYTE 15 = 63
BYTE 16 = 255
BYTE 17 = 224
BYTE 18 = 127
BYTE 19 = 255
BYTE 20 = 192
BYTE 21 = 127
BYTE 22 = 255
BYTE 23 = 128
BYTE 24 = 255
BYTE 25 = 255
BYTE 26 = 0
BYTE 27 = 255
BYTE 28 = 254
BYTE 29 = 0
BYTE 30 = 255
BYTE 31 = 252
BYTE 32 = 0
BYTE 33 = 255
BYTE 34 = 254
BYTE 35 = 0
BYTE 36 = 255
BYTE 37 = 255
BYTE 38 = 0
BYTE 39 = 127
BYTE 40 = 255
BYTE 41 = 128
BYTE 42 = 127
BYTE 43 = 255
BYTE 44 = 192
BYTE 45 = 63
BYTE 46 = 255
BYTE 47 = 224
BYTE 48 = 31
BYTE 49 = 255
BYTE 50 = 240
BYTE 51 = 7
BYTE 52 = 255
BYTE 53 = 248
BYTE 54 = 1
BYTE 55 = 255
BYTE 56 = 252
BYTE 57 = 0
BYTE 58 = 63
BYTE 59 = 254
BYTE 60 = 0
BYTE 61 = 3
BYTE 62 = 255

Supponiamo di voler visualizzare in rosso il PAC-MAN su uno sfondo giallo (il nostro mostriciattolo avrà dunque un occhio giallo); basta introdurre il seguente programmino:

```
 10 DATA 0, 3, 255, 0, 63, 254, 1, 255, 252
 20 DATA 7, 252, 248, 31, 252, 240, 63, 255, 224
 30 DATA 127, 255, 192, 127, 255, 128, 255, 255, 0
 40 DATA 255, 254, 0, 255, 252, 0, 255, 254, 0
 50 DATA 255, 255, 0, 127, 255, 128, 127, 255, 192
 60 DATA 63, 255, 224, 31, 255, 240, 7, 255, 248
 70 DATA 1, 255, 252, 0, 63, 254, 0, 3, 255
 80 POKE 2040, 56
 90 FOR I = 0 TO 62: READ A
100 POKE 56*64 + I, A: NEXT I
110 POKE 53269, 1
120 POKE 53287, 2
130 POKE 53248, 128
140 POKE 53249, 128
150 POKE 53264, 0
160 POKE 53281, 7
```

Il suo funzionamento è il seguente:

— le linee 10÷70 contengono il valore dei 63 byte che definiscono lo SPRITE
— la linea 80 fissa il puntatore di SPRITE n. 0
— le linee 90 e 100 caricano in memoria i valori che definiscono lo SPRITE
— la linea 110 accende lo SPRITE n. 0
— la linea 120 seleziona il colore rosso per lo SPRITE
— le linee 130÷150 fissano la posizione (ritorneremo su questo punto nel prossimo paragrafo)
— la linea 160 fissa il colore dello sfondo (giallo).

PAC MAN.

7) Posizione dello SPRITE

Dopo aver definito gli SPRITE dal punto di vista geometrico e del colore, è necessario posizionarli sullo schermo e possibilmente farli muovere.
La posizione di ogni SPRITE è determinata da un'ascissa X e da un'ordinata Y. Poiché la risoluzione di visualizzazione lungo l'asse X è di 320, l'ascissa potrà dunque assumere i valori compresi tra 0 e 319. Poiché X è definito da una parola di 9 bit sono autorizzati valori compresi fra 0 e 511.
Invece, poiché la risoluzione di visualizzazione lungo l'asse verticale è di 200, una parola di 8 bit è sufficiente per definire l'ordinata Y e fornisce 256 posizioni diverse.
Per ognuno degli SPRITE, X verrà dunque definito da:

— un registro di posizione lungo l'asse X (8 bit)
— un bit addizionale, posto nel registro comune agli 8 SPRITE, che è il bit più significativo della posizione lungo l'asse X

Invece Y sarà definito da:

— un registro di posizione lungo l'asse Y (8 bit).

La tavola che segue fornisce gli indirizzi di tutti i registri che permettono di posizionare gli SPRITE.

Indirizzo	Descrizione	SPRITE N°
53 248 = $D000	Posizione secondo l'asse X	0
53 249 = $D001	Posizione secondo l'asse Y	0
53 250 = $D002	Posizione secondo l'asse X	1
53 251 = $D003	Posizione secondo l'asse Y	1
53 252 = $D004	Posizione secondo l'asse X	2
53 253 = $D005	Posizione secondo l'asse Y	2
53 254 = $D006	Posizione secondo l'asse X	3
53 255 = $D007	Posizione secondo l'asse Y	3
53 256 = $D008	Posizione secondo l'asse X	4
53 257 = $D009	Posizione secondo l'asse Y	4
53 258 = $D00A	Posizione secondo l'asse X	5
53 259 = $D00B	Posizione secondo l'asse Y	5
53 260 = $D00C	Posizione secondo l'asse X	6
53 261 = $D00D	Posizione secondo l'asse Y	6
53 262 = $D00E	Posizione secondo l'asse X	7
53 263 = $D00F	Posizione secondo l'asse Y	7
53 264 = $D010	Posizione secondo l'asse Y (Bit 8 = bit più significativo	0/1/2/3/4/5/6/7

Va notato che la posizione di uno SPRITE viene sempre catalogata in rapporto all'angolo in alto a sinistra della matrice di punti 24 x 21 che lo definisce (quale che sia il contenuto dello SPRITE).
Il problema del posizionamento di uno SPRITE non è così semplice come sembra, per questo dobbiamo affrontarlo entrando nei dettagli. Poiché è più semplice, affronteremo per primo il problema del posizionamento secondo l'asse Y.

Il registro della posizione secondo l'asse Y possiede 8 bit; questo, come abbiamo già detto, permette di ottenere 255 differenti posizioni. Vedremo che, nonostante la risoluzione sia di 200 punti solamente, esse saranno tutte utilizzate.
Il controllore video fissa a 50 il valore minimo di Y che permette a uno SPRITE (di dimensioni normali o raddoppiate) di apparire totalmente nella parte alta dello schermo. Forniamo alcuni valori caratteristici:

— 9: valore minimo di Y che autorizza uno SPRITE, la cui dimensione sia raddoppiata in questa direzione, ad apparire sulla parte alta dello schermo (è visualizzata una riga di punti)
— 30: idem come sopra, ma per uno SPRITE di dimensioni normali
— 50: valore minimo di Y permette di visualizzare interamente uno SPRITE sullo schermo
— 208: valore massimo di Y che autorizza uno SPRITE, la cui dimensione sia moltiplicata per due in questa direzione, a restare interamente visibile nella parte bassa dello schermo
— 229: idem come sopra, ma per uno SPRITE di dimensioni normali
— 250: valore minimo di Y che permette di fare uscite uno SPRITE nella parte bassa dello schermo.

Nel nostro esempio di PAC-MAN, il valore di Y è uguale a 128. Se si desidera far spostare il nostro "mostriciattolo" dalla parte alta alla parte bassa dello schermo, basta semplicemente abolire la linea e aggiungere la linea seguente:

```
170 FOR I = 29 TO 250: POKE 53249, I: NEXT I
```

Affrontiamo ora il problema del posizionamento dello SPRITE sull'asse orizzontale.
Abbiamo visto che questa posizione è definita da 9 bit:

— 8 sono dati dal registro della posizione lungo l'asse X
— il 9° è situato in un registro che contiene i bit più significativi delle posizioni lungo l'asse X di ogni SPRITE (se questo bit vale "zero", allora $X \leq 255$, se vale "uno" allora $X \geq 256$).

Questo ci concede 512 posizioni diverse di cui in realtà ne vengono utilizzate solo 320. Considereremo separatamente il caso di SPRITE

di dimensione normale e il caso di SPRITE con dimensione raddoppiata lungo l'asse X.
Per gli SPRITE di dimensione normale, è semplice:

— cominciano ad apparire a sinistra sullo schermo quando X = 1
— scompaiono totalmente a destra per X = 344.

Ad esempio, per fare attraversare lo schermo da destra a sinistra al nostro PAC-MAN, basta sopprimere le linee 130 e 150 e aggiungere le linee seguenti:

```
170 FOR I = 0 TO 344
180 POKE 53264, INT(I/256)
190 POKE 53248, I – 256*INT(I/256)
200 NEXT I
```

Passiamo ora al caso degli SPRITE di dimensione estesa.
Normalmente ci vorrebbe un valore negativo di X per farli scomparire totalmente alla sinistra dello schermo. E' quanto avviene in realtà, ma bisogna utilizzare una notazione in complemento a due (i valori superiori o uguali a 256 sono considerati negativi e i valori inferiori o uguali a 255 sono considerati positivi: ad esempio, si avrà —1 = 511).
Si ottengono allora i seguenti risultati:

— uno SPRITE comincia ad apparire a sinistra sullo schermo per X = 489
— quando X aumenta fino a 511, lo SPRITE si sposta verso destra
— quando X ripassa per lo zero e aumenta fino al valore 343, lo SPRITE continua a spostarsi verso destra (scompare totalmente per X = 344).

Ad esempio, per far muovere sullo schermo il nostro PAC-MAN di dimensione estesa da sinistra a destra, si può:

— sostituire la linea 130 con:

```
130 POKE 53277, 1: POKE 53271, 1
```

(questo raddoppia la sua dimensione in entrambe le direzioni)
— sopprimere la linea 150
— aggiungere le seguenti istruzioni:

```
170 FOR I = 488 TO 511
180 POKE 53264, INT(I/256)
190 POKE 53248, I - 256*INT(I/256): NEXT
200 FOR I = 0 TO 344
210 POKE 53264, INT(I/256)
220 POKE 53248, I - 256*INT(I/256): NEXT
```

8) Priorità SPRITE-SFONDO

All'inizio di questo paragrafo abbiamo visto che è possibile rappresentare uno SPRITE in sovraimpressione sullo sfondo dello schermo oppure fare scomparire lo sfondo dietro di lui. Si tratta di un problema di priorità SPRITE-SFONDO.
Questa priorità è programmabile per ognuno degli 8 SPRITE per mezzo del registro d'indirizzo 53275 (= $D01B).
Se un bit di questo registro assume il valore 1, lo SPRITE corrispondente è visualizzato in sovraimpressione rispetto alle informazioni visualizzate sullo schermo.
Se un bit di questo registro assume il valore 0, lo SPRITE corrispondente nasconde ogni informazione che si trova dietro di lui.
Riprendiamo ancora una volta l'esempio del nostro PAC-MAN. Le linee di programma da 10 a 100 restano inalterate.

Battete ora le istruzioni seguenti:

```
110 POKE 53269, 1
120 POKE 53287, 12
130 POKE 53249, 128
140 POKE 53281, 7
150 FOR I = 1044 TO 2004 STEP 40
160 POKE I, 83: NEXT
170 POKE 53275, 1
180 FOR I = 489 TO 511
190 POKE 53264, INT(I/256)
200 POKE 53248, I - 256*INT(I/256): NEXT
210 FOR I = 0 TO 344
220 POKE 53264, INT(I/256)
230 POKE 53248, I - 256*INT(I/256): NEXT
```

Ora fate girare il programma: PAC-MAN deve scomparire dietro la riga di "cuori" visibile sullo schermo (la visualizzazione avviene in sovraimpressione).
Se ora sostituire la linea 170 con:

```
170 POKE 53275, 0
```

e fate girare nuovamente il programma, PAC-MAN deve mascherare completamente quella riga.

9) Priorità SPRITE-SPRITE

Abbiamo visto che è possibile visualizzare sullo schermo contemporaneamente 8 SPRITE. E' quindi facile che essi si incrocino.
Contrariamente alla priorità SPRITE-SFONDO, che è programmabile per ognuno degli SPRITE, la priorità SPRITE-SPRITE è fissata dal controllore video.
Lo SPRITE n. 0 ha la priorità più alta, il che significa che esso verrà visualizzato in sovraimpressione sopra qualsiasi altro SPRITE con cui si incontri.
Lo SPRITE n. 7 ha la priorità più bassa, il che significa che esso scomparirà dietro ogni altro SPRITE con cui si incontri.
La priorità va dunque diminuendo dallo SPRITE n. 0 allo SPRITE n. 7.
Questa possibilità offerta dal Commodore 64 permette di creare dei giochi che danno il senso della profondità spaziale.

10) Collisione SPRITE-SFONDO

Allo scopo di permettere la creazione di giochi con figure animate, gli SPRITE sono in grado di rilevare il momento in cui entrano in collisione con le informazioni visualizzate sullo sfondo dello schermo.
Per definizione si verifica una collisione SPRITE-SFONDO quando una parte "accesa" (bit di valore 1) dello SPRITE incontra una qualsiasi altra informazione (ad esempio un carattere).
Le eventuali collisioni SPRITE-SFONDO sono rivelate in un registro speciale chiamato "registro di collisione SPRITE-SFONDO" e situato all'indirizzo 53279 (=$D01F). Ogni SPRITE dispone di un bit in questo registro.

Quando avviene una collisione tra lo SPRITE e lo sfondo dello schermo, il bit corrispondente del registro viene posizionato sul valore 1. Questo bit conserva tale valore 1 fino a quando il microprocessore non vada a leggere il contenuto del registro (con una PEEK).
Quando avviene una lettura, questo bit viene automaticamente azzerato, permettendo così la rivelazione di una nuova collisione.
Illustreremo quanto abbiamo detto con un esempio: ritorniamo al nostro PAC-MAN e alla riga di "cuori". Ci proponiamo di rivelare una collisione tra lo SPRITE e questa riga e di farlo ritornare all'estremità sinistra dello schermo se questa avviene.
Basta per questo riaggiustare il programma già visto con le linee seguenti:

```
230 POKE 53248, I – 256*INT(I/256)
240 IF PEEK(53279) < > 0 THEN GO TO 260
250 NEXT I
260 GO TO 180
```

11) Collisione SPRITE-SPRITE

Uno SPRITE, così come è in grado di rivelare una collisione con lo sfondo dello schermo, è anche in grado di rivelare una collisione con un altro SPRITE.
Si parlerà in tal caso di collisione SPRITE-SPRITE.
Essa funziona con caratteristiche analoghe a quella SPRITE-SFONDO ma utilizza il registro di indirizzo 53278(=$D01E).
Quando uno SPRITE ne incontra un altro, i bit corrispondenti di questo registro assumono il valore 1 e restano in tale valore fino a che il registro non subisca un'operazione di lettura.
Il mezzo per rivelare delle collisioni SPRITE-SFONDO e SPRITE-SPRITE è di esaminare periodicamente (in pratica ad intervalli molto vicini) i registri corrispondenti e di verificare il loro valore.
Un inconveniente legato e questo metodo è il grande consumo di tempo di macchina che limita enormemente la velocità dell'animazione nel corso di un gioco. Fortunatamente esiste un altro metodo, molto più efficace, che descriveremo qui sotto.

12) Le interruzioni

Invece di andare continuamente a verificare il valore assunto da un bit o da un registro, può essere più semplice se potessimo essere avvisati in modo automatico quando avviene una collisione. Il principio di funzionamento può essere il seguente: quando avviene una collisione, un bit di un registro specializzato assume il valore 1 e il controllore video interrompe il microprocessore (se ne viene autorizzato in precedenza) per fargli eseguire un programma speciale. Tutto questo si chiama interruzione (interrupt).
Va notato che le interruzioni sono strettamente legate al funzionamento della macchina e sono accessibili solo in linguaggio Assembler. Tuttavia ci soffermeremo brevemente su questo argomento.
Sul controllore video esistono due registri specializzati di indirizzo 53273 (=$D019) e 53274(=$D01A).
Il primo è un indirizzo di stato dedicato alle interruzioni (Interrupt Status Register) di cui esamineremo solo i bit n. 1 e n. 2.
Il bit n. 1 assume il valore 1 quando avviene una collisione SPRITE-SFONDO e resta in questo stato finché non è azzerato (facendo POKE 53273,2).
Il bit n. 2 assume il valore 1 quando avviene una collisione SPRITE-SPRITE e resta in questo stato finché non viene azzerato (facendo POKE 53273,4).
Il secondo registro permette di autorizzare le interruzioni (Interrupt Enable Register). Quando un bit di questo registro assume il valore 1, viene autorizzata l'interruzione corrispondente. Invece se un bit di questo stesso registro assume il valore 0, l'interruzione corrispondente è impedita e non avviene nulla (il programma principale continua a svolgersi regolarmente).
Ma su questo argomento non entreremo in ulteriori dettagli.
A questo punto vi sarete resi conto che le diverse tappe della programmazione degli SPRITE sono lunghe e noiose. Per agevolare queste operazioni sono disponibili degli "Editori di SPRITE", cioè del software che permette, fra l'altro, di memorizzare automaticamente i byte che definiscono uno SPRITE a partire dal suo disegno sullo schermo, conservarlo su cassetta, ecc.
Abbiamo terminato con questo lungo capitolo sul controllore video. Dopo aver parlato delle immagini passiamo ora ai suoni con la descrizione di un altro circuito altrettanto complesso e interessante di quello che abbiamo appena illustrato.

4
Il sintetizzatore di suoni

4.1 Generalità

Un'altra interessante caratteristica del Commodore 64 è la presenza di un vero sintetizzatore musicale.
Contrariamente alla maggior parte degli altri computer, non si tratta semplicemente di un generatore di note musicali; è possibile infatti fare veramente della musica, tanto più che l'uscita Audio del Commodore 64 può essere collegata ad un'amplificatore di un impianto ad alta fedeltà.
I risultati non sono certamente paragonabili a quelli dei sintetizzatori professionali, ma restano oltremodo interessanti. Tutte le funzioni che descriveremo in questo capitolo sono contenute in un unico circuito integrato, il controllore audio mod. 6581, chiamato SID (Sound Interface Device). Proprio come il controllore video che abbiamo analizzato in dettaglio nelle pagine precedenti, il controllore audio è stato progettato per essere facilmente programmato da parte dell'utilizzatore. Come vedremo nelle pagine che seguono, la generazione di note musicali, di rumore, il filtraggio avvengono in maniera completamente trasparente per l'utilizzatore.
Non dovrete dunque preoccuparvi di programmare i 25 registri presenti su questo controllore.
I registri sono situati tra gli indirizzi 54272 (=$D400) e 54296 (=$D418) e permettono di accedere anche alle seguenti caratteristiche:

— tre generatori di suono che producono frequenze comprese fra 0 e 4 kHz
— quattro forme di segnale
— tre generatori di inviluppo
— modulazione
— possibilità di sincronizzazione degli oscillatori.

Alcune di queste possibilità saranno forse sconosciute a chi non abbia familiarità con la musica elettronica. Ma non preoccupatevi, descriveremo tutto nei dettagli nelle pagine che seguono.

4.2 Il generatore di suoni

Come abbiamo appena detto, il controllore possiede tre distinti generatori di suoni, che possono funzionare separatamente o simultaneamente. E' dunque possibile creare dei pezzi musicali semplici (a una voce) o complessi (a più voci). Sul Commodore 64 possono essere trascritte vere partiture musicali. Descriveremo ora le diverse tappe necessarie per arrivare a generare una sequenza di note isolate. Per fare questo abbiamo bisogno di un minimo di conoscenza del controllore.

1) Inizializzazione dei registri interni

Prima di cominciare a fare della musica, è necessario inizializzare tutti i registri interni del SID per non ottenere sonorità strane o non desiderate. Per fare questo basta caricare il valore 0 nei 25 registri. Si scrive:

```
10 FOR I = 54272 TO 54296
20 POKE I, 0
30 NEXT I
```

Una volta compiuta questa operazione preliminare, siamo pronti per iniziare. Prima di tutto esaminiamo il ruolo e il funzionamento dei registri utilizzati per la generazione delle note.

2) Principio di funzionamento dei generatori di suono

Ogni generatore è in grado di produrre suoni di andamento, intensità e frequenza variabile. Questi parametri sono controllabili grazie a sette registri diversi per ogni voce. Un registro addizionale, comune a tutte, permette di regolare l'intensità del suono.

Gli indirizzi e le funzioni di questi registri sono forniti nella tavola che segue.

Indirizzo	Voce	Funzione
54 272 = $D400	1	Frequenza dell'oscillatore: byte meno significativo
54 273 = $D401	1	Frequenza dell'oscillatore: byte più significativo
54 274 = $D402	1	Larghezza dell'impulso: byte meno significativo
54 275 = $D403	1	Larghezza dell'impulso: byte più significativo (solo bit 0 → 3)
54 276 = $D404	1	Registro di controllo
54 277 = $D405	1	Generatore di inviluppo: ATTACK-DECAY
54 278 = $D406	1	Generatore di inviluppo: SUSTAIN-RELEASE
54 279 = $D407	2	Frequenza dell'oscillatore: byte meno significativo
54 280 = $D408	2	Frequenza dell'oscillatore: byte più significativo
54 281 = $D409	2	Larghezza dell'impulso: byte meno significativo
54 282 = $D40A	2	Larghezza dell'impulso: byte più significativo (solo bit 0 → 3)
54 283 = $D40B	2	Registro di controllo
54 284 = $D40C	2	Generatore di inviluppo: ATTACK-DECAY
54 285 = $D40D	2	Generatore di inviluppo: SUSTAIN-RELEASE
54 286 = $D40E	3	Frequenza dell'oscillatore: byte meno significativo
54 287 = $D40F	3	Frequenza dell'oscillatore: byte più significativo
54 288 = $D410	3	Larghezza dell'impulso: byte meno significativo
54 289 = $D411	3	Larghezza dell'impulso: byte più significativo (solo bit 0 → 3)
54 290 = $D412	3	Registro di controllo
54 291 = $D413	3	Generatore di inviluppo: ATTACK-DECAY
54 292 = $D414	3	Generatore di inviluppo: SUSTAIN-RELEASE
54 296 = $D418	1-2-3	Regolazione del volume (bit 0 → 3) e selezione dei filtri

Solo 22 dei 25 registri sono stati considerati. Descriviamo ora il loro funzionamento e come programmarli.

● *Frequenza dell'oscillatore*

Essa è determinata da una parola di 16 bit (byte meno significativo e byte più significativo) e varia in modo lineare in funzione del valore assunto da tale parola.
La frequenza reale di uscita dell'oscillatore si può calcolare facilmente utilizzando la formula che segue:

FREQUENZA = 0,06097*VALORE

VALORE rappresenta la parola binaria di 16 bit contenuta nei due registri di selezione della frequenza. I valori da caricare nei due registri saranno dunque i seguenti:

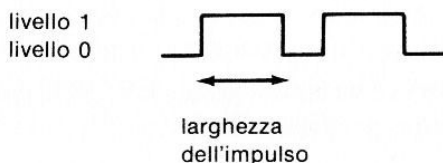
HF = INT (FREQUENZA/256)
e
LF = FREQUENZA—256* HF

In Appendice forniamo una tavola che permette di conoscere i valori di HF e LF per le note musicali.
Va notato che questo circuito di controllo audio permette di generare 8 ottave con una frequenza massima di 4 kHz. Per esempio, per generare il LA da 440 Hz basta comporre:

```
POKE 54 272, 49
POKE 54 273, 28
```

per il generatore di suono n. 1.

● *Larghezza dell'impulso*

Nel caso in cui si scelga un segnale ad impulsi, è possibile determinare, per mezzo di due registri, la larghezza dell'impulso. Infatti un segnale impulsivo ha la forma seguente:

La larghezza dell'impulso è determinata da una parola di 12 bit (i 4 bit più significativi del byte più significativo non vengono utilizzati) e varia in modo lineare con il valore assunto da quest'ultima. Questo valore può variare tra 0 e 4095, consentendo quindi ben poche variazioni. In realtà quello che si determina è il rapporto fra la durata del segnale quando si trova al livello 1 e il valore 4095 (che corrisponde a un segnale continuo costantemente a livello 1). Si ha dunque:

RAPPORTO = VALORE/4095

VALORE corrisponde alla parola binaria di 12 bit presente nei due registri.
Se VALORE = 0 si avrà un rapporto nullo, cioè un segnale costantemente a 0.
Se VALORE = 4095 si avrà un rapporto uguale a 1, cioè un segnale costantemente a 1 (5 V).
Se VALORE = 2048 si avrà un rapporto uguale a 1/2, cioè un segnale quadrato periodico.
Per avere VALORE = 2048 basta battere:

```
POKE 54274, 0
POKE 54275, 8
```

per il generatore di suono n. 1

● *Registro di controllo*

Questo registro consente sia di selezionare la forma del segnale d'uscita (quadrata, triangolare o a dente di sega) sia di programmare diverse funzioni come la sincronizzazione o la modulazione. Questo registro comprende 8 bit che illustreremo ad uno ad uno.
— Bit 0: questo bit è chiamato "gate" (porta) e permette di controllare il funzionamento del generatore di inviluppo per ogni controllore di suono. Ritorneremo più avanti sulla nozione di generatore di inviluppo, ma possiamo già dire fin da subito che esso permette di determinare il timbro e l'intonazione di un segnale facendone variare l'inizio e la fine. Quando questo bit vale 1, viene inizializzato il

ciclo che permette di generare una nota. Quando tale bit passa a 0, avviene la fine del ciclo e la nota musicale si estingue.
Per generare una nota si scrive perciò:

```
POKE 54276, 1
```

mentre per estinguerla si scrive:

```
POKE 54276, 0
```

— Bit 1: questo bit, detto di "sincronizzazione", permette appunto di sincronizzare la frequenza fondamentale dell'oscillatore n. 1 con la frequenza fondamentale dell'oscillatore n. 3, al fine di ottenere effetti speciali. Per selezionare questa modalità di funzionamento, conviene fissare la frequenza dell'oscillatore n. 3 (in generale inferiore a quella dell'oscillatore n. 1). E' poi possibile, facendo variare la frequenza dell'oscillatore n. 1, ottenere delle armoniche della frequenza selezionata dall'oscillatore n. 3.
Può sembrare un po' complicato: il metodo migliore è di provare sul calcolatore.
Per selezionare questa modalità di funzionamento basta scrivere l'istruzione che segue:

```
POKE 54276, 2
```

Va notato che quando il bit 1 del registro di indirizzo 54283 (relativo alla voce n. 2) è posizionato sul valore 1, avviene la sincronizzazione dell'oscillatore n. 2 con l'oscillatore n. 1. Analogamente quando il bit 1 del registro di indirizzo 54290 vale 1, l'oscillatore n. 3 è sincronizzato con l'oscillatore n. 2.
— Bit 2: questo bit permette di ottenere quella che viene definita una modulazione. Quando vale 1 permette di sostituire in uscita i segnali triangolari dell'oscillatore n. 1 con una combinazione degli oscillatori n. 1 e n. 3.
Facendo variare la frequenza dell'oscillatore n. 1 in rapporto a quella del n. 3 è possibile ottenere effetti speciali: gong, campane, ecc.
Perché questa modalità funzioni correttamente è necessario che l'oscillatore n. 1 sia programmato per fornire segnali triangolari,

mentre la frequenza dell'oscillatore n. 3 deve essere diversa da zero. Per selezionare questa modalità, basta battere:

```
POKE 54276, 4
```

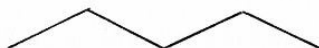
Nel caso di registro di indirizzo 54283 (voce n. 2) si ha la modulazione degli oscillatori n. 2 e n. 1. Analogamente nel caso del registro di indirizzo 54290 (voce n. 3) si ha modulazione degli oscillatori n. 3 e n. 2.

— Bit 3: questo è un bit di "test". Quando vale 1, l'oscillatore corrispondente è spento, insieme al generatore di rumore e al generatore di impulsi. Questo stato di cose cambia quando vale 0. Questo bit viene utilizzato per collaudare il controllore, ma può anche servire a sincronizzare un oscillatore con un evento esterno. Per posizionarlo nel valore 1, basta scrivere:

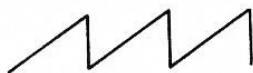
```
POKE 54276, 8 per la voce 1
POKE 54283, 8 per la voce 2
POKE 54290, 8 per la voce 3
```

— Bit 4: questo bit permette di selezionare una forma d'onda triangolare, se posizionato a 1. Questo segnale è caratterizzato da una sonorità di tipo "flautato", molto dolce. Si scrive:

```
POKE 54276, 16 per la voce 1
POKE 54283, 16 per la voce 2
POKE 54290, 16 per la voce 3
```

L'andamento del segnale è il seguente:

— Bit 5: questo bit permette di selezionare una forma di segnale a dente di sega, se posizionato a 1. Questo segnale è caratterizzato da una sonorità più brillante, analoga a quella degli "ottoni". Si scrive:

```
POKE 54276, 32 per la voce 1
POKE 54283, 32 per la voce 2
POKE 54290, 32 per la voce 3
```

L'andamento del segnale è il seguente:

— Bit 6: questo bit permette di selezionare delle forme d'onda ad impulsi. Le armoniche prodotte da questo segnale permettono di ottenere delle sonorità molto brillanti e "nasali". Queste armoniche possono essere modificate cambiando la larghezza degli impulsi mediante i due registri disponibili per tale scopo, che abbiamo già descritto.
Per selezionare questo segnale, si batte:

```
POKE 54276, 64 per la voce 1
POKE 54283, 64 per la voce 2
POKE 54290, 64 per la voce 3
```

Va notato che è possibile ottenere degli effetti di "phasing" cambiando in tempo reale la larghezza degli impulsi.

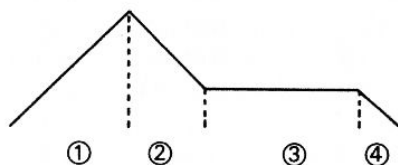
— Bit 7: questo bit, quando assume il valore 1, permette di ottenere un segnale di "rumore". Si tratta in effetti di un segnale casuale che varia alla frequenza dell'oscillatore corrispondente.
Tale tipo di segnale è utile per ottenere rumori di esplosioni, vento, onde, ecc...
Per ottenerlo, bisogna battere:

```
POKE 53276, 128 per la voce 1
POKE 53283, 128 per la voce 2
POKE 53290, 128 per la voce 3
```

● *Generatore di inviluppo*

Il ruolo del generatore di inviluppo è per così dire di determinare l'"intonazione" del suono fornito dal controllore.
Diversamente dai soliti microcomputer che forniscono dei suoni sempre al medesimo livello, con il Commodore è possibile far variare il modo con cui iniziano e finiscono.
Un suono è caratterizzato dai seguenti quattro parametri:

1. ATTACK (attacco): caratterizza la velocità con la quale un suono passa dal livello zero al livello di cresta
2. DECAY (decadimento): caratterizza la velocità con la quale un suono passa dal valore massimo (di cresta) al livello sonoro medio
3. SUSTAIN (mantenimento): caratterizza il livello sonoro medio del segnale
4. RELEASE (rilascio): caratterizza la velocità con la quale un suono passa dal livello di mantenimento (SUSTAIN) al livello zero.

Queste quattro fasi possono essere rappresentate dallo schema che segue.

Questi quattro parametri sono codificati per mezzo di quattro bit ciascuno; questo spiega la presenza di due registri per ogni generatore di suoni, utilizzati dal generatore di inviluppo. ATTACK, DECAY, SUSTAIN e RELEASE sono codificati con valori compresi tra 0 e 15.
Il primo registro (54277, per il generatore di suoni n. 1) contiene i parametri ATTACK (4 bit più significativi) e DECAY (4 bit meno significativi).
Il secondo registro(54278, per il generatore di suoni n. 1) contiene i parametri SUSTAIN (4 bit più significativi) e RELEASE (4 bit meno significativi).
La tavola che segue fornisce i valori dei tempi di attacco (ATTACK) e di discesa (DECAY, RELEASE) del segnale in funzione del valore caricato nei registri.

Valore	ATTACK	DECAY/RELEASE
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Supponiamo, ad esempio, di voler ottenere i parametri seguenti:

— tempo di salita: ATTACK = 16 ms
— tempo di discesa al livello medio: DELAY = 114 ms
— livello medio: SUSTAIN = 5 ms
— tempo di rilascio: RELEASE = 750 ms.

Si avrà dunque:

ATTACK = 2
DECAY = 4
SUSTAIN = 5
RELEASE = 9

(valori caricati nel registro di controllo del generatore di inviluppo)

Basta dunque battere per la voce n. 1):

```
POKE 54277, 2*16 + 4
POKE 54278, 5*16 + 9
```

Va notato che un valore nullo per SUSTAIN dà un livello sonoro nullo, mentre un valore uguale a 15 dà un livello sonoro medio uguale all'ampiezza massima (determinata dal registro di controllo del volume di indirizzo 54296 che descriveremo più oltre). L'uscita del generatore di suoni resta al livello determinato dal valore di SUSTAIN fintantoché il bit 0 ("gate") del registro di controllo vale 1. Quando questo bit passa a 0, comincia il ciclo di RELEASE e il segnale sonoro decresce fino a diventare nullo.

● *Registro di controllo del volume*

Questo registro è posto all'indirizzo 54296(=$D418).
Serve a controllare simultaneamente il volume dei tre generatori di suono. Inoltre è utilizzato per selezionare i filtri presenti sul controllore di suono. Ma di questo parleremo più avanti. Il livello sonoro può essere regolato per mezzo dei 4 bit meno significativi di questo registro (bit da 0 a 3).
Questo permette di selezionare dei valori compresi fra 0 e 15: al valore 0 corrisponde un livello sonoro nullo, mentre al valore 15 corrisponde il livello sonoro massimo. La variazione avviene in modo lineare.

3) Uso dei generatori di suoni

A questo punto abbiamo terminato con la descrizione dei registri di base del controllore di suoni. Essi permettono di accedere subito alla generazione di note isolate. Esamineremo ora qualche esempio e vedremo come una modifica, anche piccola, del contenuto di certi registri cambia la durata o il timbro dei suoni.
Consideriamo il programma che segue.

```
 10 FOR I = 54272 TO 54296
 20 POKE I, 0: NEXT I
 30 POKE 54277, 15: POKE 54278, 0
 40 POKE 54296, 15
 50 READ MF, BF, DU
 60 IF HF < 0 THEN GO TO 160
 70 POKE 54273, HF: POKE 54272, BF
 80 POKE 54276, 17
 90 FOR I = 1 TO DU: NEXT
100 POKE 54276, 16: FOR I = 1 TO 30: NEXT
110 GO TO 50
```

```
120 DATA 33, 135, 128, 33, 135, 128, 33, 135, 128, 37, 162, 128, 42, 62, 250,
37, 162, 250
130 DATA 33, 135, 128, 42, 62, 128, 37, 162, 128, 37, 162, 128,
33, 135, 250, -1, -1, -1
160 END
```

Questo programma funziona così:

— Le linee 10 e 20 permettono di inizializzare tutti i registri interni del controllore.
— La linea 30 permette di fissare i diversi parametri del generatore di inviluppo (ATTACK = 0, DECAY = 15, SUSTAIN = 0, RELEASE = 0). Se andate a rivedere il paragrafo che tratta del generatore di inviluppo, potete vedere che il tempo di salita della nota sarà breve (2 ms). Poi il decadimento della nota è molto lieve (tempo di discesa di 240).
  Il parametro SUSTAIN non ha importanza in questo caso, poiché il livello sonoro non ha mai il tempo di raggiungere il suo valore medio. Il parametro RELEASE = 0 permette di ottenere una rapida discesa (6 ms) della nota.
— La linea 40 regola il volume al suo livello massimo.
— La linea 50 va a leggere i diversi parametri caratteristici di una nota, cioè la componente ad alta frequenza (registro di indirizzo 54273), la componente a bassa frequenza (registro di indirizzo 54272) e la durata.
— La linea 60 permette di arrestare lo svolgimento del programma.
— La linea 70 cambia la frequenza della nota musicale nei registri previsti per tale scopo (indirizzi 54272 e 54273).
— La linea 80 permette di far partire la nota musicale per mezzo del registro di controllo:
  • Il bit 0 ("gate") assume il valore 1 per far iniziare la generazione del suono.
  • Il bit 4 permette di selezionare la forma d'onda triangolare.
— La linea 90 provoca un ritardo corrispondente alla durata della nota (mentre viene eseguito questo ciclo, il bit 0 vale 1).
— Alla linea 100 il bit 0 del registro di controllo viene azzerato per permettere l'inizio del ciclo di RELEASE. Poiché il parametro vale zero, l'istruzione della nota dura 6 ms. Il ciclo di ritardo che si trova in questa linea permette di separare le note tra loro.
— La linea 110 permette di ricominciare il ciclo per un'altra nota.
— Le linee da 120 a 150 contengono le informazioni corrispondenti alle note musicali desiderate.

Se dopo aver introdotto il programmino, battete RUN, riconoscerete le prime note di una canzoncina ben nota.
Notate la dolcezza delle note che otteniamo. Pensiamo ora a fare qualche modifica. Sostituite la linea 80 con POKE 54276,33 e la linea 100 con:

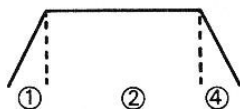
```
POKE 54276, 32: FOR I = 1 TO 30: NEXT
```

Questo permette di sostituire i segnali di forma triangolare con segnali a dente di sega. Se fate girare il programma sentirete qualcosa di molto più brillante.
Sostituendo ora queste due linee con:

```
 80 POKE 54276, 65
100 POKE 54276, 64: FOR I = 1 TO 30: NEXT
```

otterrete dei segnali di forma impulsiva. Forse ricorderete che nel caso di questa forma d'onda è possibile programmare il suo rapporto ciclico per mezzo dei registri di indirizzo 54274 e 54275. Aggiungete dunque al programma le seguenti istruzioni:

```
65 POKE 54274, 50: POKE 54275, 0
```

e fatelo girare di nuovo. Otterrete una sonorità più nasale; tuttavia modificando il valore contenuto nel registro di indirizzo 54275 è possibile ottenere delle sonorità contemporaneamente brillanti e profonde.
Ricordatevi che sono utilizzati solo i bit da 0 a 3 del registro di indirizzo 54275.
Cerchiamo di ottenere una modifica dell'intonazione dei suoni generati che agisca sul generatore di inviluppo.
Nell'esempio riportato, l'andamento dell'inviluppo era il seguente:

— il parametro ATTACK è uguale a 0 (tempo di attacco di 2 ms)
— il parametro DECAY è uguale a 15 (non esiste decadimento)

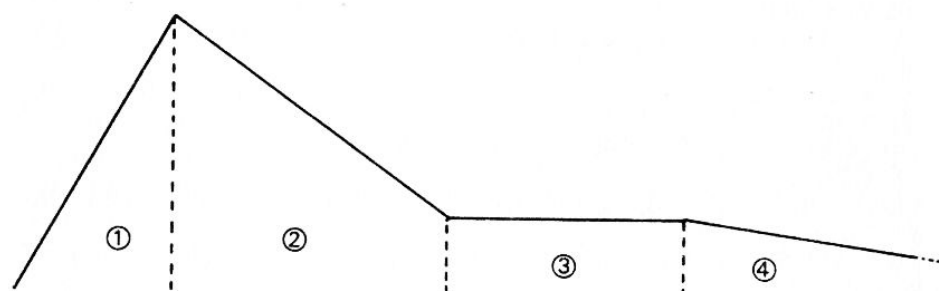
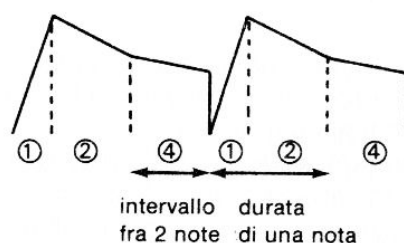
— il parametro SUSTAIN è uguale a 0 (non è utilizzato proprio per il valore del parametro DECAY)
— il parametro RELEASE è uguale a 0 (tempo di discesa di 6 ms)

Cambiando l'andamento di questo inviluppo, vi sarà possibile ottenere delle sonorità molto diverse: violino, tromba, ecc., come con un vero sintetizzatore musicale.
Cerchiamo quindi di ottenere una sonorità analoga a quella del violino, con note dolci e prolungate.
Il programmino che segue, derivato dal precedente, permette di ottenere una sonorità di questo tipo:

```
 10 FOR I = 54272 TO 54296
 20 POKE I, 0: NEXT
 30 POKE 54277, 90: POKE 54278, 94
 40 POKE 54296, 15
 50 READ HF, BF, DU
 60 IF HF < 0 THEN GO TO 160
 70 POKE 54273, HF: POKE 54272, BF
 80 POKE 54276, 33
 90 FOR I = 1 TO DU: NEXT
100 POKE 54276, 32: FOR I = 1 TO 90: NEXT
110 GO TO 50
120 DATA 33, 135, 128, 33, 135, 128, 33, 135, 128
130 DATA 37, 162, 128, 42, 62, 250, 37, 62, 250
140 DATA 33, 135, 128, 42, 62, 128, 37, 162, 128
150 DATA 37, 162, 128, 33, 135, 250, -1, -1, -1
160 END
```

I concetti su cui si basa il funzionamento di questo programma sono già stati esaminati. L'unica differenza riguarda i valori caricati nei registri di controllo del controllore di suoni.
In particolare i parametri del generatore di inviluppo hanno i seguenti valori:

ATTACK = 0101 = 5
DECAY = 1010 = 10
SUSTAIN = 0101 = 5
RELEASE = 1110 = 14

L'andamento dell'inviluppo è dunque il seguente:

In realtà la nota non corrisponde esattamente all'andamento schematizzato, poiché una sua parte ha una durata fissata dal parametro DU, mentre l'intervallo tra due note è determinato dal ciclo FOR I = TO 90 : NEXT.
Il grafico che segue precisa meglio la concatenazione delle diverse note.

Cerchiamo ora di produrre una sonorità simile a quella di un piano: le note devono essere più corte e secche. A questo scopo è necessario ricorrere a dei segnali con forma d'onda a impulsi. Il programma che segue, che deriva sempre dall'esempio precedente, illustra come fare.

```
10 FOR I = 54272 TO 54296
20 POKE I, 0: NEXT
30 POKE 54275, 3: POKE 54274, 0
40 POKE 54277, 12: POKE 54278, 0
50 POKE 54296, 15
60 READ HF, BF, DU
70 IF HF < 0 THEN GO TO 170
80 POKE 54273, HF: POKE 54272, BF
90 POKE 54276, 65
100 FOR I = 1 DU: NEXT
110 POKE 54276, 64: FOR I = 1 TO 50: NEXT
120 GO TO 60
130 DATA 33, 135, 128, 33, 135, 128, 33, 135, 128, 37, 162, 128, 42, 62, 250,
37, 162, 250
140 DATA 33, 135, 128, 42, 62, 128, 37, 162, 128, 37, 162, 128, 33, 135, 250,
-1, -1, -1
170 END
```

La linea 30 permette in questo caso di selezionare il rapporto ciclico dei segnali di tipo impulsivo. Tale rapporto in questo caso ha il valore 3 x 256/4096 = 0.188. Per ottenere note brevi e secche i parametri del generatore di inviluppo assumono i seguenti valori:

ATTACK = 0000 = 0
DECAY = 1100 = 12
SUSTAIN = 0
RELEASE = 0000 = 0

I tempi di salita e di discesa sono dunque molto brevi (ATTACK = RELEASE = 0). Il valore scelto per DECAY permette di avere un decadimento relativamente lento.
Abbiamo fin qui descritto alcuni metodi per ottenere delle sonorità vicine a quelle di strumenti conosciuti. Ben inteso, è possibile creare delle sonorità particolari: basta dare libero corso alla fantasia.
In ogni caso i diversi parametri, come:

— tipo di segnale (triangolare, dente di sega, impulsi)
— andamento dell'inviluppo (parametri ATTACK, DECAY, SUSTAIN, RELEASE)
— intensità del suono
— lunghezza della nota

dovranno essere inizialmente predeterminati in funzione di quanto desiderate ottenere e poi aggiustati "ad orecchio".

4) Uso a più voci

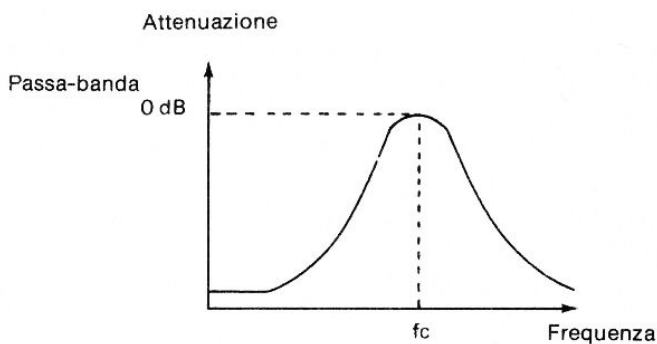
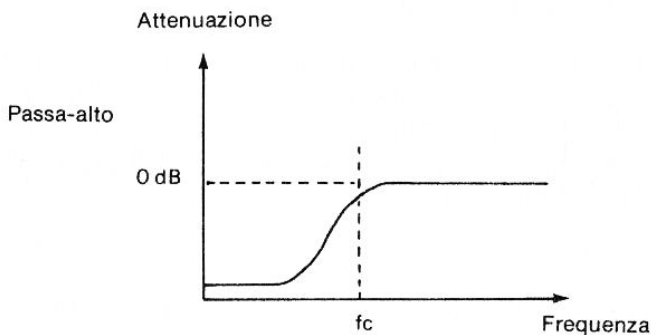
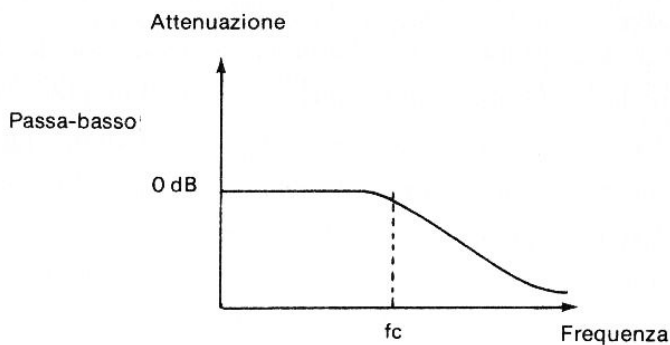
Finora abbiamo considerato la generazione di note musicali per mezzo di un solo generatore di suoni.
Ma poiché nel Commodore 64 ne esistono tre, tutti identici, è evidente che è possibile usarli contemporaneamente. La loro programmazione diventa però tanto più difficile quanto più alto è il numero delle voci.
Un punto particolarmente importante è la sincronizzazione tra i diversi canali musicali se si vuole ottenere qualcosa di accettabile. A questo scopo ogni misura musicale dovrà essere suddivisa in un numero di parti sufficienti per tener conto dei cambiamenti che intervengono in ognuna delle tre voci. Le frequenze relative a ciascuna nota e a ciascuna voce potranno allora essere immagazzinate in una tabella dimensionata in modo opportuno. Non è il caso di entrare nei dettagli di tutto questo, poiché la programmazione dipende enormemente dal brano musicale che si desidera ottenere. Quanto abbiamo detto e diremo dà tutte le informazioni necessarie all'ottenimento di partiture musicali complesse.

4.3 I filtri

Abbiamo visto che ognuno dei generatori di suono presenti sul Commodore 64 è in grado di generare segnali di andamento triangolare, a dente di sega, a impulsi o segnali di rumore.
Ognuno di questi segnali è caratterizzato da armoniche differenti che sono alla base di tonalità differenti.
Modificando le armoniche di questi segnali è dunque possibile ottenere nuove sonorità.
Prima di andare avanti, riepiloghiamo però brevemente quanto c'è da sapere sulle armoniche.
Quando si suona una nota, quest'ultima è caratterizzata da una frequenza fondamentale (quella che si può programmare per mezzo dei registri di controllo) e da un certo numero di armoniche, che hanno una frequenza multipla di quella fondamentale.

Le forme d'onda della frequenza fondamentale e delle frequenze armoniche sono di tipo sinusoidale e il segnale risultante è ottenuto facendo la loro somma ponderata per mezzo di certi coefficienti moltiplicativi.
Questi coefficienti determinano l'importanza di ogni armonica nel segnale in uscita.
Per esempio, un segnale triangolare avente una frequenza fondamentale di 1000 Hz comporterà delle armoniche da 2000 Hz, 3000 Hz, ecc. aventi delle ampiezze fisse. La somma di questi diversi segnali darà il triangolo descritto in precedenza.
Il ruolo dei filtri sarà di diminuire o sopprimere tutte o solo alcune delle armoniche per ottenere un segnale, con andamento e dunque sonorità differenti.
Come avete già capito, l'accesso a questi filtri avviene per mezzo di registri presenti sul controllore di suono. Essi sono in numero di quattro e permettono di accedere a dei filtri di tipo: passa-basso, passa-banda, passa-alto.
Ricordiamo che dei filtri di tipo ideale hanno rispettivamente le seguenti proprietà:

— un filtro passa-basso lascia passare le frequenze inferiori alla sua frequenza di taglio e "taglia" tutte le frequenze superiori
— un filtro passa-alto lascia passare tutte le frequenze superiori alla sua frequenza di taglio e "taglia" tutte le frequenze inferiori
— un filtro passa-banda lascia passare le frequenze vicine alla sua frequenza di taglio e taglia tutte le alte.

In realtà i filtri non seguono queste proprietà teoriche e il "taglio" non è mai netto. Possiamo rappresentare le caratteristiche dei diversi filtri con i diagrammi che seguono.

I registri che permettono di controllare i filtri sono posti tra gli indirizzi 54293=$D415 e 54296=$D418.
Le funzioni di questi registri sono date nella tavola che segue.

Indirizzo	Funzione
54 293 = $D415	Frequenza di taglio del filtro: 3 bit meno significativi
54 294 = $D416	Frequenza di taglio del filtro: 8 bit più significativi
54 295 = $D417	Risonanza del filtro/Controllo del filtraggio
54 296 = $D418	Selezione dei filtri/Volume

● *Registri d'indirizzo 54293 e 54294*

I primi due registri permettono di accedere a una parola di 11 bit. I tre bit meno significativi sono posti nel registro di indirizzo 54293 e gli otto bit più significativi sono nel registro di indirizzo 54294. Questa parola permette di programmare una frequenza di taglio compresa approssimativamente fra 30 Hz e 12 kHz.

● *Registro di indirizzo 54295*

— Bit 0: quando questo bit vale zero, il segnale proveniente dal generatore di suono n. 1 è trasmesso senza filtri in uscita. Quando assume il valore uno, questo segnale passa attraverso il filtro selezionato e il segnale risultante viene trasmesso in uscita.
— Bit 1: questo bit funziona come il bit 0, ma questa volta per il generatore di suono n. 2.
— Bit 2: questo bit funziona come il bit 0, ma questa volta per il generatore di suono n. 3.
— Bit 3: questo bit viene utilizzato per un ingresso audio esterno. Infatti esiste sul connettore Audio/Video DIN a 5 piedini un ingresso destinato ad un segnale audio esterno. Quando questo bit assume il valore zero, il segnale proveniente da questo ingresso è trasmesso all'uscita senza modifiche. Invece quando questo bit assume il valore uno, il segnale è filtrato per mezzo del filtro selezionato e poi trasmesso in uscita.
— Bit 4÷7: questi bit permettono di programmare la risonanza del

filtro selezionato. In pratica, quanto maggiore è la risonanza tanto maggiore è l'amplificazione della componente del segnale alla frequenza di taglio del filtro. Quando i bit da 4 a 7 valgono zero, non c'è risonanza. Quando assumono il valore uno, la risonanza è massima.
Esistono perciò 16 valori possibili e la variazione ha un andamento lineare.

- *Registro d'indirizzo 54296*

Il ruolo dei bit da 0 a 3 di questo registro è già stato descritto prima (regolazione del volume del suono). I bit da 4 a 7 permettono di selezionare i vari filtri:

— Bit 4: quando assume il valore uno, viene selezionato il filtro passa-basso
— Bit 5: quando assume il valore uno, viene selezionato il filtro passa-banda
— Bit 6: quando assume il valore uno, viene selezionato il filtro passa-alto
— Bit 7: quando assume il valor uno, il segnale emesso dal generatore di suono n. 3 non è più trasmesso all'uscita anche se continua ad oscillare.

Questo permette ad esempio di ottenere delle modulazioni. Il filtro può essere anche cortocircuitato per non influenzare il segnale che esce dal generatore n. 3 mettendo a zero il bit 2 del registro di indirizzo 54295.

Vediamo ora come i vari filtri agiscono sul segnale ottenuto in uscita. A questo scopo riprendiamo il primo esempio del paragrafo riguardante "l'uso dei generatori di suono", avendo cura di selezionare un andamento dei segnali ricco di armoniche; ad esempio, dei segnali a dente di sega (la linea 80 sarà dunque sostituita da POKE 54276,33 e alla linea 100 si avrà POKE 54276,32).
Aggiungiamo ora le seguenti istruzioni:

```
23 POKE 54293, 0: POKE 54294, 180
25 POKE 54295,1: POKE 54296, 31
```

Sopprimendo poi la linea 40, si ottiene una sonorità molto meno brillante e molto più dolce.
La linea 23 permette di determinare la frequenza di taglio del filtro passa-basso che è selezionato alla linea 25.
Il filtraggio sopprime una parte armonica del segnale iniziale e tende a farlo avvicinare a un segnale sinusoidale "puro".
Abbiamo visto l'effetto di un filtraggio passa-basso su un segnale a dente di sega. Gli effetti più sorprendenti si ottengono però quando si utilizzano i generatori di rumore integrati nel controllore di suono, proprio a causa della loro ricchezza in armoniche. Anche in questo caso sarà necessario fare delle prove per ricercare nuove sonorità.
Va notato che i filtri possono essere utilizzati simultaneamente (ad esempio filtro passa-alto e filtro passa-basso).

4.4 Programmazione avanzata

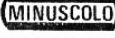
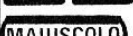
In questo capitolo abbiamo illustrato le tecniche di base che permettono di generare delle note, dei gruppi di note e anche delle vere e proprie partiture musicali. Abbiamo invece lasciato un po' da parte le possibilità di rumore offerte dal controllore audio.
Per fare questo dobbiamo parlare di un altro registro di questo controllore.
Tale registro è posto all'indirizzo 54299 e può essere soltanto letto. Esso fornisce gli 8 bit più significativi dell'ampiezza digitale del generatore di suono n. 3.
Infatti il circuito di controllo audio lavora in maniera digitale (dunque con dei numeri) e può generare un segnale analogico (segnale sonoro) grazie all'intermediazione di uno speciale convertitore chiamato convertitore digitale-analogico.
Il contenuto di questo registro passa dunque da 0 a 255 e inversamente ad un ritmo determinato dalla frequenza dell'oscillatore n. 3 e dall'andamento dei segnali selezionati.
Ad esempio, un segnale ad impulsi permetterà il passaggio diretto da 0 a 255, mentre un segnale triangolare genererà tutti i passi intermedi.
Se si scegliesse un rumore, il contenuto di questo registro sarebbe casuale.
Il ruolo principale di questo registro è la modulazione.
E' ad esempio possibile generare un segnale, tipo quello di una sirena, controllando la frequenza dell'oscillatore n. 1 per mezzo dell'ampiezza

di uscita dell'oscillatore n. 3. Quest'ultimo dovrà essere inibito (bit 7 del registro di indirizzo 54296 posizionato al valore uno).
Il programma che segue illustra tutto questo:

```
 10 FOR I = 54272 TO 54296
 20 POKE I, 0: NEXT
 30 POKE 54275, 1
 40 POKE 54276, 65
 50 POKE 54278, 240
 60 POKE 54286, 12
 70 POKE 54290, 16
 80 POKE 54296, 143
 90 FI = 10000
100 FOR I = 1 GO TO 500
110 F2 = F + 20*PEEK(54299)
120 HF = INT(F2/256): LF = F2 - 256*HF
130 POKE 54272, LF: POKE 54273, HF
140 NEXT
```

Il funzionamento del programma è molto semplice:

— si seleziona un segnale a impulsi per l'oscillatore n. 1 (linee 30, 40, 50)
— si seleziona un segnale triangolare per l'oscillatore n. 3 (linee 60, 70)
— viene inibita l'uscita dell'oscillatore n. 3 e il volume viene messo al massimo (linea 80)
— il valore della frequenza dell'oscillatore n. 1 viene determinato dalle istruzioni della linea 110 in funzione del valore dell'ampiezza dell'oscillatore n. 3.

Oltre al registro di indirizzo 54299, che permette di conoscere il valore dell'ampiezza del segnale emesso dal generatore di suono n. 3, esiste un altro registro (che può essere solamente letto), il quale fornisce l'uscita del generatore di inviluppo n. 3. Per mezzo di questo registro è possibile controllare sia la frequenza di un oscillatore sia la frequenza di taglio di un filtro.
Per determinare questa descrizione del controllore di suono, dobbiamo menzionare la presenza di due ultimi registri. Essi si trovano agli indirizzi 54297=$D419 e 54298=$D41A. Permettono di conoscere il valore digitalizzato di un potenziometro (X o Y) collegato al circuito.

Il primo registro permette di conoscere il valore del potenziometro X, mentre il secondo permette di conoscere il valore del potenziometro Y. Questi registri contengono dei valori compresi tra 0 e 255. Il valore 0 corrisponde a una resistenza minima, mentre il valore 255 corrisponde alla massima resistenza. Questi registri possono essere utilizzati per leggere il valore delle "paddle" collegate alle prese per i giochi (Control Port n. 1 e Control Port n. 2).
A questo punto abbiamo terminato sia con questo capitolo sia con questo libro dedicato all'uso del Commodore 64.
Come avete certamente capito, questa macchina offre notevoli potenzialità dal punto di vista della grafica e della generazione dei suoni. In ogni caso vi invitiamo a fare molte prove, ad uscite dai sentieri battuti, poiché questo a nostro avviso è il modo migliore di conoscere un microcomputer e divertirsi con lui.
Questo vuole essere però anche un appuntamento per il libro successivo che approfondirà altri aspetti del Commodore 64 in particolare le periferiche e le funzioni di ingresso e uscita.

Appendici

TAVOLA DEI CODICI ASCII E CHR$

PRINT	CHR$	PRINT	CHR$	PRINT	CHR$	PRINT	CHR$
	0	CRSR ↓	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
Disa-bilita SHIFT C=	8		25	*	42	;	59
Abilita SHIFT C=	9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	−	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
MINUSCOLO	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

PRINT	CHR$	PRINT	CHR$	PRINT	CHR$	PRINT	CHR$
D	68		97		126	GRIGIO 3	155
E	69		98		127	PUR	156
F	70		99		128	CRSR	157
G	71		100	ARANCIO	129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	MAIUSCOLO	142		171
U	85		114		143		172
V	86		115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120	MARRONE	149		178
£	92		121	ROSSO CHIARO	150		179
]	93		122	GRIGIO 1	151		180
↑	94		123	GRIGIO 2	152		181
←	95		124	VERDE CHIARO	153		182
	96		125	AZZURRO	154		183

PRINT	CHR$	PRINT	CHR$	PRINT	CHR$	PRINT	CHR$
	184		186		188		190
	185		187		189		191

Codici da 192 a 223: sono identici ai codici da 96 a 127
Codici da 224 a 254: sono identici ai codici da 160 a 190
Codice 255 : è identico a 126

TAVOLA DEI CODICI-SCHERMO

Serie n.1	Serie n.2	Codice	Serie n.1	Serie n.2	Codice	Serie n.1	Serie n.2	Codice
@		0	Y	y	25	2		50
A	a	1	Z	z	26	3		51
B	b	2	[		27	4		52
C	c	3	£		28	5		53
D	d	4	]		29	6		54
E	e	5	↑		30	7		55
F	f	6	←		31	8		56
G	g	7	SPACE		32	9		57
H	h	8	!		33	:		58
I	i	9	"		34	;		59
J	j	10	#		35	<		60
K	k	11	$		36	=		61
L	l	12	%		37	>		62
M	m	13	&		38	?		63
N	n	14	'		39			64
O	o	15	(		40	♠	A	65
P	p	16	)		41		B	66
Q	q	17	*		42		C	67
R	r	18	+		43		D	68
S	s	19	,		44		E	69
T	t	20	–		45		F	70
U	u	21	.		46		G	71
V	v	22	/		47		H	72
W	w	23	0		48		I	73
X	x	24	1		49		J	74

Serie n.1	Serie n.2	Codice	Serie n.1	Serie n.2	Codice	Serie n.1	Serie n.2	Codice
	K	75			100			125
	L	76			101			126
	M	77			102			127
	N	78			103			
	O	79			104			
	P	80			105			
	Q	81			106			
	R	82			107			
	S	83			108			
	T	84			109			
	U	85			110			
	V	86			111			
	W	87			112			
	X	88			113			
	Y	89			114			
	Z	90			115			
		91			116			
		92			117			
		93			118			
		94			119			
		95			120			
SPACE		96			121			
		97			122			
		98			123			
		99			124			

I codici da 128 a 255 sono l'inversione video dei codici da 0 a 127.

TAVOLA DELLE NOTE MUSICALI

Nota	Frequenza	Valore decimale	Byte più significativo	Byte meno significativo
do	16.3	268	1	12
do diesis	17.3	284	1	28
re	18.4	301	1	45
re diesis	19.4	318	1	62
mi	20.5	337	1	81
fa	21.8	358	1	102
fa diesis	23.1	379	1	123
sol	24.4	401	1	145
sol diesis	25.9	425	1	169
la	27.5	451	1	195
la diesis	29.1	477	1	221
si	30.9	506	1	250
do	32.7	536	2	24
do diesis	34.6	568	2	56
re	36.7	602	2	90
re diesis	38.8	637	2	125
mi	41.2	675	2	163
fa	43.7	716	2	204
fa diesis	46.2	758	2	246
sol	49.0	803	3	35
sol diesis	51.9	851	3	83
la	55.0	902	3	134
la diesis	58.2	955	3	187
si	61.7	1 012	3	244
do	65.4	1 072	4	48
do diesis	69.3	1 136	4	112
re	73.4	1 204	4	180
re diesis	77.7	1 275	4	251
mi	82.4	1 351	5	71
fa	87.3	1 432	5	152
fa diesis	92.5	1 517	5	237
sol	98	1 607	6	71
sol diesis	104	1 703	6	167
la	110	1 804	7	12
la diesis	117	1 911	7	119
si	123	2 025	7	233
do	131	2 145	8	97
do diesis	139	2 273	8	225
re	147	2 408	9	104
re diesis	156	2 551	9	247
mi	165	2 703	10	143
fa	175	2 864	11	48
fa diesis	185	3 034	11	218
sol	196	3 215	12	143
sol diesis	208	3 406	13	78
la	220	3 608	14	24
la diesis	233	3 823	14	239
si	247	4 050	15	210

Nota	Frequenza	Valore decimale	Byte più significativo	Byte meno significativo
do	262	4 291	16	195
do diesis	277	4 547	17	195
re	294	4 817	18	209
re diesis	311	5 103	19	239
mi	330	5 407	21	31
fa	349	5 728	22	96
fa diesis	370	6 069	23	181
sol	392	6 430	25	30
sol diesis	415	6 812	26	156
la	440	7 217	28	49
la diesis	466	7 647	29	223
si	494	8 101	31	165
do	523	8 583	33	135
do diesis	554	9 094	35	134
re	587	9 634	37	162
re diesis	622	10 207	39	223
mi	659	10 814	42	62
fa	699	11 457	44	193
fa diesis	740	12 139	47	107
sol	784	12 860	50	60
sol diesis	831	13 625	53	57
la	880	14 435	56	99
la diesis	932	15 294	59	190
si	988	16 203	63	75
do	1 047	17 167	67	15
do diesis	1 109	18 188	71	12
re	1 175	19 269	75	69
re diesis	1 245	20 415	79	191
mi	1 319	21 629	84	125
fa	1 397	22 915	89	131
fa diesis	1 480	24 278	94	214
sol	1 568	25 721	100	121
sol diesis	1 661	27 251	106	115
la	1 760	28 871	112	199
la diesis	1 865	30 588	119	124
si	1 976	32 407	126	151
do	2 093	34 334	134	30
do diesis	2 218	36 376	142	24
re	2 350	38 539	150	139
re diesis	2 489	40 830	159	126
mi	2 637	43 258	168	250
fa	2 794	45 830	179	6
fa diesis	2 960	48 556	189	172
sol	3 136	51 443	200	243
sol diesis	3 323	54 502	212	230
la	3 521	57 743	225	143
la diesis	3 730	61 176	238	248
si	3 952	64 814	253	46

TAVOLA DEI CODICI-COLORE

Codici	Colore
0	Nero
1	Bianco
2	Rosso
3	Celeste
4	Porpora
5	Verde
6	Blu
7	Giallo
8	Arancio
9	Marrone
10	Rosso chiaro
11	Grigio 1
12	Grigio 2
13	Verde chiaro
14	Azzurro
15	Grigio 3

Come usare il tuo Commodore 64

Con il libro "Come usare il tuo Commodore 64" scoprirete le possibilità più interessanti offerte dal vostro personal computer.
Vengono descritti in dettaglio e chiaramente le istruzioni e i comandi del BASIC raggruppandoli per categorie affini.
Potrete anche imparare ad accedere alle diverse modalità di visualizzazione dei dati e dei grafici e a programmare i famosi (e un po' misteriosi) SPRITE.
Inoltre un intero capitolo è dedicato alla sintesi dei suoni. Potrete così comporre le vostre musiche o generare effetti sonori particolari.
Tutti questi argomenti sono illustrati con numerosi esempi scritti in BASIC.